METADATA-BASED AND PERSONALIZED WEB QUERYING

A DISSERTATION SUBMITTED TO
THE DEPARTMENT OF COMPUTER ENGINEERING
AND THE INSTITUTE OF ENGINEERING AND SCIENCE
OF BILKENT UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

By
Selma Ayşe Özel
January, 2004

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of doctor of philosophy.

Prof. Dr. Özgür Ulusoy (Advisor)

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of doctor of philosophy.

Prof. Dr. Erol Arkun

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of doctor of philosophy.

Assoc. Prof. Dr. Nihan Kesim Çiçekli

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of doctor of philosophy.

Assist. Prof. Dr. Uğur Güdükbay

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of doctor of philosophy.

Prof. Dr. Enis Çetin

Approved for the Institute of Engineering and Science:

Prof. Dr. Mehmet B. Baray
Director of the Institute

ABSTRACT

METADATA-BASED AND PERSONALIZED WEB QUERYING

Selma Ayşe Özel
Ph.D. in Computer Engineering
Supervisor: Prof. Dr. Özgür Ulusoy
January, 2004

The advent of the Web has raised new searching and querying problems. Keyword matching based querying techniques that have been widely used by search engines, return thousands of Web documents for a single query, and most of these documents are generally unrelated to the users' information needs. Towards the goal of improving the information search needs of Web users, a recent promising approach is to index the Web by using metadata and annotations.

In this thesis, we model and query Web-based information resources using metadata for improved Web searching capabilities. Employing metadata for querying the Web increases the precision of the query outputs by returning semantically more meaningful results. Our Web data model, named "Web information space model", consists of Web-based information resources (HTML/XML documents on the Web), expert advice repositories (domain-expert-specified metadata for information resources), and personalized information about users (captured as user profiles that indicate users' preferences about experts as well as users' knowledge about topics). Expert advice is specified using topics and relationships among topics (i.e., metalinks), along the lines of recently proposed topic maps standard. Topics and metalinks constitute metadata that describe the contents of the underlying Web information resources. Experts assign scores to topics, metalinks, and information resources to represent the "importance" of them. User profiles store users' preferences and navigational history information about the information resources that the user visits. User preferences, knowledge level on topics, and history information are used for personalizing the Web search, and improving the precision of the results returned to the user.

We store expert advices and user profiles in an object relational database

management system, and extend the SQL for efficient querying of Web-based information resources through the Web information space model. SQL extensions include the clauses for propagating input importance scores to output tuples, the clause that specifies query stopping condition, and new operators (i.e., text similarity based selection, text similarity based join, and topic closure). Importance score propagation and query stopping condition allow ranking of query outputs, and limiting the output size. Text similarity based operators and topic closure operator support sophisticated querying facilities. We develop a new algebra called Sideway Value generating Algebra (SVA) to process these SQL extensions.

We also propose evaluation algorithms for the text similarity based SVA directional join operator, and report experimental results on the performance of the operator. We demonstrate experimentally the effectiveness of metadata-based personalized Web search through SQL extensions over the Web information space model against keyword matching based Web search techniques.

Keywords: metadata based Web querying, topic maps, user profile, personalized Web querying, Sideway Value generating Algebra, score management, text similarity based join.

ÖZET

METADATAYA DAYALI VE KİŞİSELLEŞTİRİLMİŞ WEB SORGULAMASI

Selma Ayşe Özel
Bilgisayar Mühendisliği, Doktora
Tez Yöneticisi: Prof. Dr. Özgür Ulusoy
Ocak, 2004

Web'in gelişimi ile beraber, bilgiye erişim ve sorgulamada yeni problemler ortaya çıkmıştır. Çoğunlukla arama motorları tarafından kullanılan anahtar söz karşılaştırmaya dayalı sorgulama yöntemleri tek bir sorgu için binlerce Web belgesi getirmekte ve bu belgelerin çoğu kullanıcıların bilgi ihtiyaçları ile ilgisiz olmaktadır. Web kullanıcılarının bilgi arama ihtiyaçlarını iyileştirmek amacına yönelik olarak, son umut verici yaklaşım Web'in metadata ve ek açıklama kullanılarak dizinlenmesidir.

Bu tezde, Web arama yeteneklerini iyileştirmek için, Web'deki bilgi kaynakları metadata kullanılarak modellenmekte ve sorgulanmaktadır. Web sorgulamasının metadata kullanılarak yapılması, daha anlamlı sorgu sonuçlarının üretilmesini sağlamaktadır. "Web bilgi uzayı modeli" adını verdiğimiz Web veri modeli, Web tabanlı bilgi kaynaklarından (Web üzerindeki HTML/XML formundaki belgelerden), uzman öneri veritabanlarından (bilgi kaynakları için alan uzmanı tarafından hazırlanmış metadatadan), ve kullanıcılarla ilgili kişiselleştirilmiş bilgiden (kullanıcıların uzmanlarla ilgili tercihleri ve konular hakkındaki bilgi seviyesini belirleyen kullanıcı profillerinden) oluşmaktadır. Uzman önerisi, yakın zamanda önerilmiş olan konu haritaları standardı doğrultusunda, konular ve konular arasındaki ilişkiler (metalink'ler) kullanılarak tanımlanmaktadır. Konular ve konular arasındaki ilişkiler, Web'deki bilgi kaynaklarının içeriğini tanımlayan metadata'yı oluştururlar. Uzmanlar, konulara, konular arasındaki ilişkilere ve bilgi kaynaklarına onların önem derecesini belirten sayısal değerler verirler. Kullanıcı profilleri kullanıcıların tercihlerini ve kullanıcıların ziyaret ettikleri bilgi kaynaklarını içeren tarihçeyi saklamaktadırlar. Kullanıcı tercihleri, konular üzerindeki bilgi seviyeleri ve Web dolaşım tarihçesi Web'deki aramayı

kişiselleştirmek ve kullanıcıya döndürülen sonucun duyarlılığını arttırmak için kullanılır.

Uzman önerileri ve kullanıcı profilleri nesneye dayalı ilişkisel veritabanında saklanmakta ve Web tabanlı bilgi kaynaklarını Web bilgi uzayı modeli kullanarak etkin şekilde sorgulayabilmek için SQL dili genişletilmektedir. SQL uzantıları, girdi önem değerlerinin çıktı kayıtlarına iletimini sağlayan yantümceleri, sorguyu durdurma koşulunu tanımlayan yantümceyi ve yeni işleçleri (metin benzerliğine dayalı seçim, metin benzerliğine dayalı birleşim, ve konu kapsamı) içerir. Önem değerinin iletimi ve sorguyu durdurma koşulu sorgu çıktısının sıralanmasını ve çıktı boyutunun sınırlandırılmasını sağlar. Metin benzerliğine dayalı işleçler ve konu kapsamı işleci karmaşık sorgulama olanaklarını desteklemektedir. Bu SQL eklentilerini işleyebilmek amacıyla "Yan Değer üreten Cebir" adı verilen yeni bir cebir geliştirilmiştir.

Yan değer üreten cebir tanımlandıktan sonra, metin benzerliğine dayalı yönlü birleştirme işlecinin algoritması ve bu işlecin performansı üzerine olan deneysel sonuçlar sunulmaktadır. Tüm bunlara ek olarak, Web bilgi uzayı modeli üzerinde SQL eklentileri kullanılarak yapılan metadataya dayalı kişiselleştirilmiş Web sorgulamasının etkinliği, anahtar söz karşılaştırmaya dayalı Web arama teknikleri ile karşılaştırmalı olarak gösterilmiştir.

Anahtar sözcükler: metadataya dayalı Web sorgulaması, konu haritaları, kullanıcı profili, kişiselleştirilmiş Web sorgulaması, Yan Değer üreten Cebir, değer yönetimi, metin benzerliğine dayalı birleştirme.

Acknowledgement

First of all, I am deeply grateful to my supervisor Prof. Dr. Özgür Ulusoy, for his invaluable suggestions, support, and guidance during my graduate study, and for encouraging me a lot in my academic life. It was a great pleasure for me to have a chance of working with him.

I would like to address my special thanks to Prof. Dr. Gültekin Özsoyoğlu and Prof. Dr. Z. Meral Özsoyoğlu, for their revisions and support, which invaluably contributed to this thesis.

I would like to thank Prof. Dr. Erol Arkun, Assoc. Prof. Dr. Nihan Kesim Çiçekli, Assist. Prof. Dr. Uğur Güdükbay, and Prof. Dr. Enis Çetin for reading and commenting this thesis. I would also like to acknowledge the financial support of Bilkent University, TÜBİTAK under the grant 100U024, and NSF (of the USA) under the grant INT-9912229.

I am grateful to my colleague İ. Sengör Altıngövde, for his cooperation during this study. I would also like to thank my friends Rabia Nuray, Berrin-Cengiz Çelik for their friendship and moral support.

Above all, I am deeply thankful to my parents, my husband Assist. Prof. Dr. A. Alper Özalp and also his parents, who supported me in each and every day. Without their everlasting love and encouragement, this thesis would have never been completed.

To my family

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Due to the property of being easily accessible from everywhere, the World Wide Web has become the largest resource of information that consists of huge volumes of data of almost every kind of media. However, due to the large size of the Web data, finding relevant information on the Web becomes like searching for a needle in a haystack.

The growing amount of information on the Web has lead to the creation of new information retrieval techniques, such as high quality human maintained indices e.g., Yahoo!, and search engines. At the moment, 85% of the Internet users are reported to be using search engines [43] because of the fact that human maintained lists cover only popular topics, are subjective, expensive to build and maintain, slow to improve, and can not cover all topics. Search engines, on the other hand, are based on automatic indexing of Web pages with various refinements and optimizations (such as ranking algorithms that make use of links, etc). Yet, the biggest of these engines cannot cover more than 40% of the available Web pages [10], and even worse some advertisers intentionally mislead them to gain people's attention [17]. Consequently, the need for better search services to retrieve the most relevant information is increasing, and to this end, a more recent and promising approach is indexing the Web by using metadata and annotations. After the proposal of the XML (eXtensible Markup Language) [16] as a data exchange format on the Web, several frameworks such as semantic Web effort [12],

RDF (Resource Description Framework) [73], and topic maps [13, 14, 71] to model the Web data in terms of metadata objects have been developed. Metadata based indexing increases the precision of the query outputs by returning semantically more meaningful query results.

Our goal in this thesis is to exploit metadata (along the lines of recently proposed topic maps), XML and the DBMS (Database Management System) perspective to facilitate the information retrieval for arbitrarily large Web portions. We describe a "Web information space" data model for metadata-based modeling of a *subnet*[1]. Our data model is composed of:

- Web-based *information resources* that are XML/HTML documents.
- Independent *expert advice repositories* that contain domain expert-specified description of information resources and serve as metadata for these resources. Topics and metalinks are the fundamental components of the expert advice repositories. Topics can be anything (keyword, phrase, etc.) that characterizes the data at an underlying information resource. Metalinks are relationships among topics.
- *Personalized information* about users, captured as user profiles, that contain users' preferences as to which expert advice they would like to follow, and which to ignore, etc., and users' knowledge about the topics that they are querying.

We assume that our data model can be stored in a commercial object relational DBMS, and we extend the SQL (Structured Query Language) with some specialized operators (e.g., topic closure, similarity based selection, similarity based join, etc.) to query the Web resources through the Web information space model. We illustrate the metadata-based querying of the Web resources with an example.

[1]We make the practical assumption that the modeled information resources do not span the Web; they are defined within a set of Web resources on a particular domain, which we call *subnet*s, such as the TREC Conference series sites [80], or the larger domain of Microsoft Developers Network sites [61].

Example 1.1 Assume that a researcher wants to see the list of all papers and their sources (i.e., ps/pdf/HTML/XML files containing the full text of the papers) which are located at the DBLP (Database and Logic Programming) Bibliography [51] site, and are prerequisite papers for understanding the paper "DMQL: A Data Mining Query Language for Relational Databases" by Jiawei Han et al. [38]. Presently, such a task can be performed by extracting the titles of all papers that are cited by Han et al.'s paper and intuitively eliminating the ones that do not seem like prerequisites for understanding the original paper. Once the user manually obtains a list of papers (possibly an incomplete list), he/she retrieves each paper one by one, and examines them to see if they are really prerequisites or not. If the user desires to follow the prerequisite relationship in a recursive manner, then he/she has to repeat this process for each paper in the list iteratively. Clearly, the overall process is time inefficient. Instead, let's assume that an expert advice (i.e., metadata) is provided for the DBLP Bibliography site. In such a metadata model, "research paper", "DMQL: A Data Mining Query Language for Relational Databases", and "J. Han" would be designated as topics, and *Prerequisite* and *ResearchPaperOf* are relationships among topics (referred to as topic metalinks). For each topic, there would be links to Web documents containing "occurrences" of that topic (i.e., to DBLP Bibliography pages), called topic sources. Then, the query can be formulated over this metadata repository, which is typically stored in an object-relational DBMS, and the query result is obtained (e.g., the prerequisite paper is "Mining Sequential Patterns" by Agrawal et al. [3]). Figure 1.1 shows the metadata objects employed in this example for the DBLP Bibliography Web resources.

In Example 1.1, we assume that an expert advice repository on a particular domain (e.g., DBLP Bibliography site) is provided by a domain expert either manual or in (semi)automated manner. It is also possible that different expert advice repositories may be created for the same set of Web information resource(s) to express varying viewpoints of different domain experts. Once it is formed, the expert advice repository captures valuable and lasting information about the Web resources even when the information resource changes over time. For instance, the expert advice repository given in Example 1.1 stores the *ResearchPaperOf*

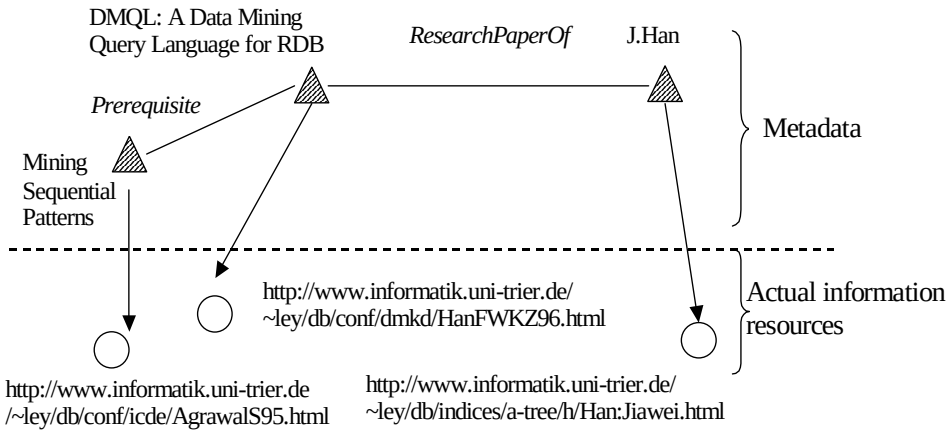

Figure 1.1: Metadata model for DBLP Bibliography domain defined by an expert.

relationship between two topics, "J. Han" and his research paper, which is a valuable and stable information even when the corresponding DBLP Bibliography resources for the paper or author are not available any more.

As we deal with querying Web resources, and ranking of query outputs appears frequently in Web-based applications, we assume that experts assign *importance scores* (or *sideway values*) to the instances of topics, metalinks, and sources that appear in their advices. We employ the scores of expert advice objects to generate scores for query output objects, which are then used to rank the query output.

Example 1.2 Consider the expert advice and the query given in Example 1.1. Assume that the query poser wants to see the list of top-10 topic important research papers that are prerequisites to the paper "DMQL: A Data Mining Query Language for Relational Databases" by J. Han et al. In this case, the importance scores assigned by the expert to the papers and *Prerequisite* metalink instances are used to rank the query output. Let's assume that the expert assigns scores 1 to the paper "DMQL: A Data Mining Query Language for Relational Databases", 0.9 to "Mining Sequential Patterns", and 0.7 to the *Prerequisite* metalink instance ("Mining Sequential Patterns" is *Prerequisite* to "DMQL: A Data Mining Query Language for Relational Databases"). Then, the *revised* topic importance score of the paper "Mining Sequential Patterns" is 1*0.9*0.7 which equals 0.63. And, the output of the query is ranked with respect to the revised importance scores. We discuss score assignment and score management issues in

more detail in subsequent chapters of this thesis.

Our SQL extensions, designed to facilitate metadata-based Web querying, also allow approximate text similarity comparisons as the majority of the Web consists of text documents, and experts assign arbitrary names to the topics in the metadata that they generate for a subnet. To support text similarity comparisons in the queries, we develop text similarity based selection and join operators which are not provided in standard SQL. Text similarity based selection is used when the query poser does not know the exact names for the topics that he/she is looking for. Text similarity based join operator is employed to integrate and query multiple expert advice databases from different sources.

In this thesis, we study text similarity based join operator in more detail, since the join operator is more crucial than the selection operator, has more application domains, and the optimization techniques that we benefit from during the processing of the join operator is also applicable to the selection operator. We propose an algorithm for text similarity based join operator and show through experimental evaluations that our algorithm is more efficient than the previously proposed algorithms in the literature in terms of number of tuple comparisons and disk accesses made during the join operation. We also incorporate some short cut evaluation techniques from the Information Retrieval domain, namely Harman [39], quit [63], continue [63], and maximal similarity filter [69] heuristics, for reducing the amount of similarity computations performed during the join operation.

Finally, we experimentally evaluate the performance of the metadata-based Web querying by running some test queries over two different metadata databases. One of the expert advice repositories contains metadata about famous horror novelist Stephen King and his books, the other includes metadata for all research papers located at the DBLP Bibliography site. The Stephen King metadata is created manually by a domain expert by browsing hundreds of documents about Stephen King on the Web. The DBLP Bibliography metadata, on the other hand, is generated semi-automatically by a computer program. For both metadata databases, we demonstrate that the proposed Web data model and

the SQL extensions, used for querying the Web, lead to higher quality results compared to the results produced by a typical keyword-based searching. We also observe that employing user preferences and user knowledge during the query processing further improves the precision of the query outputs.

1.1 Summary of the Contributions

The main contributions of this thesis can be summarized as follows:

- A metadata model making use of XML and topic maps paradigm is described for Web resources.
- A framework to express user profiles and preferences in terms of these metadata objects is presented.
- An algebra and query processing algorithms that extend SQL for querying expert advice repositories with some specific operators (similarity based selection, join, etc.) are presented.
- Query processing algorithms that employ short-cut evaluation techniques from the Information Retrieval domain for the similarity based join operator are proposed.
- An experimental evaluation of metadata-based search as compared to keyword based search is provided. In the experiment, we employ two expert advice databases; one of them is manually created and the other one is semi-automatically generated. This also allows us to compare the query output precision of manually generated metadata with semi-automatically generated one.

1.2 Organization of the Thesis

We provide the background and related work in Chapter 2 where the related standards XML, RDF, and topic maps, the Web query languages, metadata-based Web querying efforts (e.g., semantic Web), and top-k query processing issues are summarized. Chapter 3 is devoted to the description of our Web information space model and the discussion on practical issues to create and maintain expert advice repositories and user profiles. We present SQL extensions along with new operators and their query processing algorithms in Chapter 4. In Chapter 5, we discuss the text similarity based join operator in more detail, and experimentally evaluate all the join algorithms presented in this thesis. Chapter 6 includes the performance evaluation experiments of the metadata-based Web search. We conclude and point out future research directions in Chapter 7. Finally, we give the extended SQL statements of the queries that are employed in the performance evaluation experiments in Appendix A.

Chapter 2

Background and Related Work

In our metadata-based Web querying framework, we first exploit the DTDs of information resources on the Web that are XML files, to generate the metadata database along the lines of the topic map standard. We then store the metadata in an object relational DBMS, and extend the SQL with specialized operators (i.e., textual similarity based selection, textual similarity based join, and topic closure) and score management facilities to query the metadata database, and provide effective Web searching.

As background and related work to our study, we summarize the related standards XML, topic maps, and RDF in Section 2.1, discuss the Web query languages and other metadata-based querying proposals in Section 2.2, and present previous score management proposals along with ranked query evaluation in Section 2.3.

2.1 Related Standards

2.1.1 XML

As the size of the World Wide Web has been increasing extraordinarily, the abilities of HTML have become insufficient for the requirements of Web technology.

HTML is limited for the new Web applications, because HTML does not allow users to specify their own tags or attributes in order to semantically qualify their data, and it does not support the specification of deep structures needed to represent database schemas or object oriented hierarchies [15]. To address these problems, the eXtensible Markup Language (XML) was developed by an XML Working Group, organized by the World Wide Web Consortium (W3C) in 1996, as a new standard that supports data exchange on the Web.

Like HTML, XML is also a subset of SGML. However, HTML was designed specifically to describe how to display the data on the screen. XML, on the other hand, was designed to describe the content of the data, rather than presentation. XML differs from HTML in three major respects. First of all, XML allows new tags to be defined at will. In XML, structures can be nested to arbitrary depth, and finally an XML document can contain an optional description of its grammar [1]. XML data is self-describing, and therefore, it is possible for programs to interpret the XML data [78].

The structure of XML documents are described by DTDs (Document Type Definition), and they could be considered as schemas for XML documents. The structure of an XML document is specified by giving the names of its elements, sub-elements, and attributes in its associated DTD [78]. DTDs are not only used for constraining XML documents, but can also be used in query optimization for XML query languages [74], and efficient storage [27] and compression [54] of XML documents.

Relational, object-relational, and object databases can be represented as XML data [1]. However, XML data has a different structure from these traditional data models in the sense that XML data is not rigidly structured and it can model irregularities that cannot be modeled by relational or object oriented data [26]. For example, in XML data, data items may have missing elements or multiple occurrences of the same element; or elements may have atomic values in some data items and structured values in others; and as a result of this, collection of elements can have heterogeneous structure. In order to model and store XML data, Lore's XML data model [35], ARANEUS Data Model (ADM) [57], and

a storage language STORED [27] have been proposed. Besides, the authors in [77, 78] describe how to store XML files in relational databases. For storing XML files in relational tables, first the schemas for relational tables are extracted from the DTD of the XML files, and then each element in the XML files is inserted as one or multiple tuples to the relational tables.

2.1.2 Topic Maps

Topic maps standard is a metadata model for describing data in terms of topics, associations, occurrences and other specific constructs [13]. In other words, a topic map is a structured network of hyperlinks above an information pool [42]. In such a network, each node represents a named topic and links among them represent their relationships (associations) [72]. Thus, a topic map can be basically seen as an SGML (or XML) document in which different element types are used to represent topics, occurrences of topics and relationships between topics. In this respect, the key concepts can be defined as follows [6, 13, 71, 72]:

Topic: A topic represents anything; a person, a city, an entity, a concept, etc. For example, in the context of *computer science*, a topic might represent subjects such as "Database Management Systems", "XML", "Computer Engineering Department", or "Bilkent University" (anything about computer science). What is chosen as topic highly depends on the needs of the application, the nature of the information, and the uses to which the topic map will be put.

Topic Type: Every topic has one or more types, which are a typical class-instance relation and they are themselves defined as topics. Therefore, "Database Management Systems" would be a topic of type *subject*, "XML" a topic type of *markup language* or *subject*, "Computer Engineering Department" of type *academic department*, and "Bilkent University" of type *university*. Topic types *subject*, *markup language*, *academic department*, and *university* are also topics.

Topic Name: Each topic has one or more names. The topic map standard [42] includes three types of names for a topic that are *base name*, *display name*, and *sort name*. For the topic "Bilkent University", the base name and the sort name could be "Bilkent U.", and the display name would be "Bilkent University".

Topic Occurrence: A topic occurrence is a link to a resource (or more than one resource) that is relevant to the subject that the topic represents. Occurrence(s) of a topic can be an article about the topic in a journal, a picture or video depicting the topic, a simple mention of the topic in the context of something else, etc. Topic occurrences are generally outside of the topic map, and they are "pointed at" using an addressing mechanism such as XPointer. Occurrences may be of any different types (e.g., article, illustration, mention, etc.) such that each type is also a topic in the topic map, and occurrence types are supported in the topic map standard by the concept of the occurrence role.

Topic Association: An association describes the relationship between two or more topics. For instance, "XML" *is a subject in* "Database Management Systems", "Database Management Systems" *is a course in* "Computer Engineering Department", etc. Each association is of a specific association type. In the examples, *is a subject in*, *is a course in* are association types. Each associated topic plays a role in the association. In the relationship "Database Management Systems" *is a course in* "Computer Engineering Department", those roles might be *course* and *department*. The association type and association role type are both topics.

Scope and Theme: Any assignment to a topic is considered valid within certain limits, which may or may not be specified explicitly. The validity limit of such an assignment is called its *scope*, which is defined in terms of topics called *themes*. The limit of validity of the relation "Database Management Systems" *is a course in* "Computer Engineering Department" may be the fall semesters. So, the *scope* of this relation is "Fall", and the *theme* is "graduate courses".

Public Subject: This is an addressable information resource which unambiguously identifies the subject of topic in question. As an example, the public subject for the topic "XML" may be the Web address of the document [84] which defines the "XML" standard officially. Public subject for a topic is used when two or more topic maps are merged. As the topic names assigned to a topic may differ from one topic map to other, to identify whether two topics having different names are the same topics or not, their public subjects are compared.

The basic motivation behind topic maps was the need to be able to merge indexes belonging to different document collections [71]. However, topic maps are also capable of handling tables of contents, glossaries, thesauri, cross references etc. The power of topic maps as navigational tools comes from the fact that they are topic-oriented and they utilize an index which encapsulates the structure of the underlying knowledge (in terms of topics, associations and other related notions); whereas search engines simply use (full-text) index which can not model the semantic structure of the information resources over which it is constructed [67, 72]. Thus, topic maps are the solution for query posers who want fast access to selected information in a given context.

As it is mentioned above, topic maps are a kind of semantic index over the information resources, and the occurrences of topics are just links to actual information resources which are outside of the topic map. This allows a separation of information into two domains: the metadata domain (topics and associations) and the occurrence (document) domain [68, 71]. The metadata domain itself is a valuable source of information and it can be processed without regard for the topic occurrences. Thus, it is possible that different topic maps can be created over the same set of information resources, to provide different views to users [71]. Also, topic maps created by different authors (i.e., information brokers) may be interchanged and even merged. In [79] a publicly available source codes, and in [64] a commercial tool for creating and navigating topic maps are presented. Thus, an information broker can design topic maps and sell them to information provider or link them to information resources and sell them to end-users [72].

2.1.3 RDF

RDF (Resource Description Framework) [73] is another technology for processing metadata, and it is proposed by the World Wide Web Consortium (W3C). RDF allows descriptions of Web resources to be made available in machine understandable form. One of the goals of RDF is to make it possible to specify semantics for data based on XML in a standardized, interoperable manner. The basic RDF data model consists of three object types [73]:

Resources: Anything being described by RDF expressions is called resource. A resource may be a Web page (e.g., "http://cs.bilkent.edu.tr/courses/-cs351.html"), or a part of a Web page (e.g., a specific HTML or XML element within the document source), or a whole collection of pages (e.g., an entire Web site). A resource may also be an object that is not directly accessible via the Web (e.g., a printed journal).

Properties: A property is a specific characteristic or attribute used to describe a resource. Each property has a specific meaning that defines the types of resources it can describe, and its relationship with other properties.

Statements: A specific resource together with a named property and the value of that property for that resource is called an RDF statement. These three parts of a statement are called, the subject, the predicate, and the object, respectively. The object of a statement (i.e., the property value) can be another resource or it can be a literal. In RDF terms, a literal may have content that is XML markup but is not further evaluated by the RDF processor. As an example consider the sentence "Engin Demir is the creator of the resource http://cs.bilkent.edu.tr/courses/cs351.html". The subject (resource) of this sentence is http://cs.bilkent.edu.tr/courses/cs351.html, the predicate (property) is "creator" and the object (literal) is "Engin Demir".

Thus, the RDF data model provides an abstract, conceptual framework for defining and using metadata, as the topic maps data model does. And, both RDF and topic maps use the XML encoding as its interchange syntax. However, one

difference of RDF from topic maps is that RDF annotates directly the information resources; topic maps, on the other hand, create a semantic network on top of the information resources. RDF is centered on resources, while topic maps on topics [56]. Although topic maps and RDF are different standards, the main goal of both of them is the same, and current research includes the integration and interoperability of the two proposals [34, 49, 65].

2.2 Query Languages for Information Extraction from the Web

As the size and usage of the Web increase, the problem of searching the Web for a specific information becomes an important research issue. As a solution to this problem, a number of query languages (e.g., WebSQL, W3QL, WebLog, StruQL, FLORID, TMQL, RQL, etc.) have been proposed.

2.2.1 Web Query Languages

In [31], a comprehensive survey for querying the Web using database-style query languages is provided. The query languages WebSQL, W3QL, and WebLog, as their names imply were designed specifically for querying the Web.

WebSQL is a high level SQL like query language developed for extracting information from the Web [8]. WebSQL models the Web as a relational database that is composed of two virtual relations: Document and Anchor [31]. Document relation has one tuple for each document in the Web, and consists of *url*, *title*, *text*, *type*, *length*, and *modif* attributes, where *url* is the Uniform Resource Locator (URL) for the Web document and it is the primary key of the relation since URL can uniquely identify a relation; *title* is the title of the Web document, *text* is the content or whole document, *type* of a document may be HTML, Postscript, image, audio, etc., *length* is the size of the document, and *modif* is the last modification date. All attributes are character strings, and except the URL, all

other attributes can be null. Anchor relation has one tuple for each hypertext link in each document in the Web, and it consists of *base*, *href*, and *label* attributes where *base* is the URL of the Web document containing the link, *href* is the referred document, and *label* is the link description [58].

A WebSQL query consists of **select-from-where** clauses and it starts querying with a user specified URL given in the **from** clause, and follows interior, local, and/or global hypertext links in order to find the Web documents that satisfy the conditions given in the **where** clause. A hypertext link is said to be *interior* if the destination is within the source document, *local* if the destination and source documents are different but located on the same server, and *global* if the destination and the source documents are located on different servers. Arrow-like symbols are used to denote these hypertext links. For example $\mapsto$ denotes an interior link, $\rightarrow$ denotes a local link, $\Rightarrow$ represents a global link, and = is used for an empty path. Path regular expressions are formed by using these arrow-like symbols with concatenation (.), alternation (|), and repetition (*).

The below query

select d.url, d.title
from Document d **such that**
"http://www.cs.toronto.edu" = | $\rightarrow$ | $\rightarrow\rightarrow$ d
where d.title **contains** "database"

starting from the Department of Computer Science home page of the University of Toronto, lists the URL and title of each Web document that are linked through paths of length two or less containing only local links, and having the string "database" in their title.

WebSQL can also be used for finding broken links in a page, defining full text index based on the descriptive text, finding references from documents in other servers, and mining links [8].

WebOQL is another language that has been proposed for querying the Web.

Unlike WebSQL, WebOQL not only models hypertext links between Web documents, but it also considers the internal structure of the Web documents [7]. The main data structure of WebOQL is hypertree. A hypertree is a representation of a structured document containing hyperlinks. Hypertrees are ordered arc-labeled trees with two types of arcs, *internal* and *external*. Internal arcs are used to represent structured objects (Web documents) and external arcs are used to represent hyperlinks among objects. Arcs are labeled with records.

A set of related hypertrees forms a web[2]. A WebOQL query maps hypertrees or webs into other hypertrees or webs, and consists of **select-from-where** clauses. In WebOQL queries, navigation patterns are used to specify the structure of the paths that must be followed in order to find the instances for variables. Navigation patterns are regular expressions whose alphabet is the set of predicates over records. WebOQL can simulate all nested relational algebra operators, and can create and manipulate web [7].

Several other languages have also been proposed in order to query the Web. W3QL [44, 45], WebLog [50], and WQL [53] are among these query languages. W3QL and WQL are similar to WebSQL, however WebLog uses deductive rules instead of the SQL-like syntax.

StruQL [30] is a query language of STRUDEL, which is a system for implementing data intensive Web sites. A StruQL query can integrate information from multiple data sources, and produce a new Web site according to the content and structure specification given in the query.

FLORID [40, 55] is another Web query language that is based on F-logic. FLORID provides a powerful formalism for manipulating semistructured data in a Web context. However, it does not support the construction of new Webs as a result of computation; the result is always a set of F-logic objects.

All the Web query languages mentioned in this section try to model and query the Web as a whole by considering the inter document link structures, however our work is distinguished from these proposals in that we focus on querying a

[2]A web is a data structure used in WebOQL, and it consists of a set of related hypertrees.

subset of the Web on a specific domain (i.e., subnet) by employing a metadata database over the subnet.

2.2.2 Topic Maps and RDF based Query Languages

TMQL [46] is a topic map query language designed specifically to query the topic and association entities, not the topic occurrences of topic maps. TMQL is an extension of SQL in a way that it handles the topic map data structure. The input and output of a TMQL query are both topic maps. tolog [33] is another language to query the topic maps. tolog is inspried from Prolog, and it has the same querying power with TMQL. However, tolog operates on a higher level of abstraction than the TMQL, and may perform operations that would be exceedingly difficult in TMQL.

The basic idea behind the TMQL is similar to that of our work in the sense that both proposals extend the SQL to query the topic maps. The difference is that our SQL extensions are more sophisticated such that we have designed specialized operators; "text similarity based selection" and "text similarity based join" to support IR-style text similarity based operations, and "topic closure" to allow useful queries that can not be formed in any other Web querying framework. We also include score management to SQL which is not supported in TMQL. The topic map query language tolog does not support our specialized operators and score management facility too.

Semantic Web [12] is an RDF schema-based effort to define an architecture for the Web, with a schema layer, logical layer, and a query language. The Semantic Web Workshop [28] contains various proposals and efforts for adding semantics to the Web. In [56], a survey on Semantic Web related knowledge representation formalisms (i.e., RDF, topic maps, and DAML+OIL [41]) and their query languages are presented. Among those query languages, RQL is the one supporting more features than the other proposals.

RQL is developed in the context of C-Web project [22] which is an effort to

support information sharing within the specific Web communities (e.g., in Commerce, Culture, Health). The main design goals of the project include (i) creation of conceptual models (schema), which could be carried out by knowledge engineers and domain experts and exported in RDF syntax, (ii) publishing information resources using the terminology of conceptual schema, and (iii) enabling community members to query and retrieve the published information resources. The querying facilities are provided by the language RQL. RQL relies on a formal graph model that enables the interpretation of superimposed resource descriptions. It adapts the functionality of XML query languages to RDF and it extends this functionality by uniformly querying both ontology and data.

In WebSemantics (WS) system [62], an architecture is provided to publish and describe data sources for structured data on the WWW along with a language based on WebSQL [58] for discovering resources and querying their metadata. The basic ideas and motivation of C-Web project and WebSemantics are quite similar to our work, but the approaches for modeling, storing and querying the metadata differ. Our metadata model basically relies on topic maps data model which we store in a commercial object relational DBMS, and query through SQL extensions. Our specialized operators and score management functionality are not supported in C-Web and WebSemantics.

2.3 Top-k Query Processing

As we bring score management functionality to SQL in order to limit the cardinality of the output, and rank the output with respect to their score, our work is also related to the top-k query processing which has been investigated by many database researchers recently. Carey et al. performed one of the earliest studies on ranked query processing [18]. In that work, an SQL extension, "stop after" clause that enables query writers to control the query output size is proposed. After everything else specified in the query are performed, the stop after clause retains only the first n tuples in the result set. If the "order by" clause is also specified in the query, then only the first n tuples according to this ordering are

returned as the query output. In [19], more recent strategies are presented for efficient processing of the "stop after" operator.

In another related work, Chaudhuri et al. developed a technique for evaluating a top-k selection query by translating it into a single range query [21]. In that work, n-dimensional histograms are employed to map a top-k selection query consisting of n attributes to a suitable range query. Fagin et al. were also interested in finding top-k matching objects to a given query [29].

Several algorithms for top-k join operator are presented in [11, 20, 66]. The problem of optimizing and executing multi-join queries is considered in [11]. Natsev et al. examined the problem of incremental joins of multiple ranked data sets with arbitrary user-defined join predicates on input tuples [66]. It is assumed in their work that, they are given m streams of objects (relations) ordered on a specific score attribute for each object, and a set of p arbitrary predicates defined on object attributes. A valid join combination includes exactly one object from each stream subject to join predicates. Each combination is evaluated through a monotone score aggregation function, and the k join combinations having the highest scores are returned as output. Similarly, Chang et al. present an algorithm for evaluating ranked top-k queries with expensive predicates [20]. They also describe a join algorithm that outputs the top-k joined objects having the highest scores.

The text similarity based directional join operator of our work is different from all the above top-k join proposals in the sense that, it joins each tuple from one relation with k tuples from the other relation having the highest scores (similarity). The output size of the top-k join operators, on the other hand, is at most k. In our text similarity based directional join operator, we consider similarity of the join attributes as the join predicate, while the top-k join operators employ more general join predicates. Also, all the top-k query processing algorithms assume that the objects (tuples) in all relations are sorted with respect to a score value, however our join algorithm does not require input relations be sorted.

Chapter 3

Web Information Space Model

In this chapter, we present our Web information space model, which is used to provide metadata-based modeling of subnets. The Web information space model was first introduced in [5, 6]. The model is composed of information resources on the Web, expert advice repositories, and personalized information about users.

3.1 Information Resources

Information resources are Web-based documents containing data of any type such as bulk text in various formats (e.g., ascii, postscript, pdf, etc.), images with different formats (e.g., jpeg), audio, video, audio/video, etc. In this thesis, we assume that information resources are in the form of XML/HTML documents, however, our model allows any kind of media to be information resources as long as metadata about them are provided.

We name an information resource in which a particular topic occurs as *topic source*. For example, the ps/pdf document containing the full text of the paper "DMQL: A Data Mining Query Language for Relational Databases" constitutes a topic source for the topic of type PaperName and having topic name "DMQL:

A Data Mining Query Language for Relational Databases". Also, all other documents that cite this PaperName topic in ACM Portal Web site [2] constitute a topic source for this topic. For XML-based information resources, we assume that a number of topic source attributes are defined within the XML document (using XML element tags) such as **LastUpdated**, **Author**, and **MediaType** attributes, etc.

The metadata about the data contained in topic sources are stored in expert advice repositories. Also, the expert advice repository, discussed next, has an entity, called "topic source reference", which contains (partial) information about a topic source (such as its Web address, etc).

3.2 Expert Advice Model

In our Web information space model, expert advices are metadata that describe the contents of associated information resources. Each domain expert models a subnet (a set of information resources in a particular domain) in terms of

- topic entities,
- topic source reference entities, and
- metalinks (i.e., metalink types, signatures and instances).

Our expert advice model is in fact a subset of the topic map standard [42], however, we extend the standard with some additional attributes associated to topic, topic source reference and metalink entities. We discuss the similarities and differences between our expert advice model and the topic map standard wherever appropriate in the subsequent sections.

Expert advice repositories are stored in a traditional object-relational DBMS such that, there is a table for topics, topic source references, and each metalink type. We assume that, expert advice repositories are made available by the associated institutions (e.g., DBLP Bibliography Web site) to be used for sophisticated

querying purposes. Besides, independent domain experts (i.e., information brokers [72]) could also publish expert advice repositories for particular subnets on their Web sites as a (probably feed) service. We briefly discuss a semi-automated means of creating such expert advice repositories in Section 3.4, after we describe the properties of the model in detail. In [5, 47, 52], detailed discussion on creation and maintenance of expert advice repositories is provided.

3.2.1 Topic Entities

A topic entity represents anything; a person, a city, a concept, etc. as in the topic map standard discussed in Chapter 2. In our expert advice model, topic entity has **T(opic-)Name**, **T(opic-)Type**, **T(opic-)Domain** (scope), **Roles**, etc. attributes as specified in the topic map standard (see Chapter 2). In our model, topics also have the following additional attributes which are not supported in topic map standard.

T(opic-)Author attribute defines the expert (name or id or simply a URL that uniquely identifies the expert) who authors the topic.

T(opic-)MaxDetailLevel. Each topic can be represented by a topic source in the Web information resource at a different detail level. Therefore, each topic entity has a maximum detail level attribute. Let's assume that levels 1, 2 and 3 denote levels "beginner", "intermediate", and "advanced". For the "data mining" domain, for example, a source for topic "association rule mining" can be at a beginner (i.e., detail level 1) level, denoted by "Association Rule Mining^1" (e.g., "Apriori Algorithm"). Or, it may be at an advanced (say, detail level n) level of "Association Rule Mining^n" (e.g., "association rule mining based on image content"). Note the convention that topic x at detail level i is more advanced (i.e., more detailed) than topic x at detail level j when $i>j$.

T(opic-)Id. Each topic entity has a T(opic-)Id attribute, whose value is an artificially generated identifier, internally used for efficient implementation purposes, and not available to users.

T(opic-)SourceRef. Each topic entity has a T(opic-)SourceRef attribute which contains a set of Topic-Source-Reference entities as discussed in the next subsection.

T(opic-)Importance-Score. Each topic entity has a T(opic)-Importance-Score attribute whose value represents the "importance" of the topic. An importance score is a real number in the range [0, 1], and it can also take its value from the set {No, Don't-Care}. The importance score is a measure for the importance of the topic, except for the cases below.

1. When the importance value is "No", for the expert, the metadata object is rejected (which is different from the importance value of zero in which case the object is accepted, and the expert attaches a zero value to it). In other words, metadata objects with importance score "No" are not returned to users as query output.
2. When the importance value is "Don't-Care", the expert does not care about the use of the metadata object (but will not object if the other experts use it), and chooses not to attach any value to it.

Experts assign importance scores to topics in manual/semi-automated/automated manner, which is discussed in Section 3.4.1.

The attributes (**TName**, **TType**, **TDomain**, **TAuthor**) constitute a key for the topic entity. And, the **TId** attribute is also a key for topics. The topic entity that we describe in this section is very similar to the one specified in the topic map standard, however our topic entity has extra attributes (e.g., **TMaxDetailLevel**, **TImportance-Score**) which do not exist in the topic map standard, and these attributes play important role for efficient Web querying as we discuss in Chapter 4.

3.2.2 Topic Source Reference Entities

A T(opic-)S(ource-)Ref(erence), also an entity in the expert advice model, contains additional information about topic sources. This entity is similar to the

topic occurrence entity in the topic map standard; the difference is, we extend topic source reference entity with the following attributes:

Topics (set of Tid values) attribute that represents the set of topics for which the referenced source is a topic source.

Web-Address (URL) of the topic source.

Start-Marker (address) indicating the exact starting address of the topic source relative to the beginning of the information resources (e.g., http://MachineLearning.org/DataMining#Apriori).

Detail-Level (sequence of integers). Each topic source reference has a detail level describing how advanced the level of the topic source is for the corresponding topic.

Other possible attributes of topic source reference entities include **S(ource)-Importance-Score**, **Mediatype**, **Role** and **Last-Modified**.

3.2.3 Metalink Entities

Topic Metalinks represent relationships among topics. For instance, "DMQL: A Data Mining Query Language for Relational Databases" is *ResearchPaperOf* "J. Han", "Y. Fu", "W. Wang", "K. Koperski", and "O. Zaine" represents a metalink instance between a research paper and a set of authors. In topic map standard topic metalinks are called *topic associations*. As topic metalinks represent relationships among topics, not topic sources, they are "meta" relationships, hence our choice of the term "metalink". Metalinks have the following attributes which are different from the attributes of topic associations.

M(etalink-)Type represents the type of the relationship among the topics. In the example, "DMQL: A Data Mining Query Language for Relational Databases" is *ResearchPaperOf* "J. Han", "Y. Fu", "W. Wang", "K. Koperski", and "O. Zaine", the metalink type is *ResearchPaperOf*.

M(etalink-)Signature serves as a definition for a particular metalink type, and includes the name given to the metalink type and the topic types of topics that are related with this metalink type. For instance, the signature "*ResearchPaperOf*(E): research paper → SetOf (researcher)" denotes that according to the expert E, the *ResearchPaperOf* metalink type can hold between topics of types "researcher" and "research paper".

Ant(ecedent)-Id is the topic-id(s) of topic(s) that is on the left hand side of a metalink instance. For the above metalink instance, Ant-Id is the topic id for the topic "DMQL: A Data Mining Query Language for Relational Databases".

Cons(equent)-Id is the topic-id(s) of topic(s) that is on the right hand side of a metalink instance. For the above metalink instance, Cons-Id is the set of topic ids for the topics "J. Han", "Y. Fu", "W. Wang", "K. Koperski", and "O. Zaine".

Metalink entities also have other attributes such as **M(etalink-)Domain**, **M(etalink-)Id**, and **M(etalink-)Importance-Score** as described for topic entities.

There may be other metalink types. For instance, *Prerequisite* is a metalink type with the signature *Prerequisite*(E): SetOf (topic) → SetOf (topic). The metalink instance "Apriori Algorithm[2]" → *Prerequisite* "Association Rule Mining from Image Data[1]" states that "Understanding of the topic "Apriori Algorithm" at level 2 (or higher) is the prerequisite for understanding the topic "Association Rule Mining from Image Data" at level 1". Yet another metalink relationship can be the *RelatedTo* relationship that states, for example, that the topic "association rule mining" is related to the topic "clustering". *SubTopicOf* and *SuperTopicOf* metalink types together represent a topic composition hierarchy. As an example, the topic "information retrieval" is a super-topic (composed) of topics "indexing", "text similarity comparison", "query processing", etc. The topic "inverted index" is a sub-topic of "indexing" and "ranked query processing". Thus any relationship involving topics deemed suitable by an expert in the field can be a topic metalink.

3.3 Personalized Information Model: User Profiles

The user profile model maintains for each user his/her preferences about experts, topics, sources, and metalinks as well as the user's knowledge about topics. Thus, our personalized information model consists of two components: user preferences, and user knowledge.

3.3.1 User Preferences

In our Web information space model, we employ user preference specifications, along the lines of Agrawal and Wimmers [4]. The user U specifies his/her preferences as a list of Accept-Expert, T(opic)-Importance etc. statements, as shown in Example 3.1. Essentially, these preferences indicate in which manners the expert advice repositories can be employed while querying underlying information resources. In this sense, they may affect the query processing strategies for, say, a query language or a higher-level application that operates on the Web information space model.

In particular, the Accept-Expert statement captures the list of expert advice repositories (their URLs) that a user relies and would like to use for querying. Next, T(opic)-Importance and S(ource)-Importance statements allow users to specify a threshold value to indicate that only topics, or topic sources with greater importance scores than this threshold value are going to be used during query processing and included in the query outputs. Furthermore, the users can express (through Reject-T and Reject-S statements) that they don't want a topic with a particular name, type, etc., or a topic source at a certain location to be included in the query outputs, regardless of their importance scores. Finally, when there are more than one expert advice repositories it is possible that different experts assign different importance scores to the same metadata entities. In this case, the score assignments are accepted in an ordered manner as listed by the Accept-Expert statement. We illustrate user preferences with an example.

Example 3.1 Assume that we have three experts www.information-retrieval.org (E1), www.IR-research.org (E2), and www.AI-resources.org (E3). The user John-Doe is a researcher on information retrieval and specifies the following preferences:

Accept-Expert(John-Doe) = {E1, E2}
T-Importance(John-Doe) = {(E1, 0.9), (E2, 0.5)}
S-Importance(John-Doe) = {(E1, 0.5)}
Reject-T(John-Doe) = {(E2, TName= "*image*")}
Reject-S(John-Doe) = {Web-Address= www.hackersalliance.org}

We assume that the user preferences are practically stored in an object-relational DBMS, in this example; preferences are shown as a list of statements for the sake of comprehensibility. The first preference states that Prof. Doe wants to use expert advice repositories E1 and E2 to query the underlying Web resources, but not E3 (which includes metadata about irrelevant resources to his research area). The second and third clauses further constrain that only topics and sources with importance values greater than the specified threshold values should be returned as query output. For instance, a topic from repository E1 will be retrieved only if its importance score is greater than 0.9. The fourth preference expresses that Prof. Doe does not want to see any topics that include the term "image" in its name from the repository E2, as he is only interested in text retrieval issues. The fifth one forbids any resource from the site www.hackersalliance.org to be included in any query outputs. Finally, if there is a conflict in the importance scores assigned to a particular topic or source by experts E1 and E2, then, first, advices of E1 and then only non-conflicting advices from E2 are accepted. For example, assume that the topic "text compression" has the importance score 0.9 in E1 and "No" in E2. Then, the topic "text compression" is included in the query results, since the conflicting advice from E2 is not considered. As another example, assume that expert E1 assigns importance score of "Don't Care" for topic "distributed query processing" and expert E2 assigns 0.6 importance score for that topic. Then, the topic is included in the query results, given that E1 does not care whether the topic is included or not, but E2 assigns the importance score

of 0.6, which is greater than the threshold value specified in the T-Importance statement.

3.3.2 User Knowledge

For a given user and a topic, the knowledge level of the user on the topic is a certain detail level of that topic. The knowledge level on a topic cannot exceed the maximum detail level of the topic. The set U-Knowledge(U) = {(topic, detail-level-value)} contains users' knowledge on topics in terms of detail levels. While expressing user knowledge, topics may be fully defined using the three key attributes *TName*, *TType* and *TDomain*, or they may be partially specified in which case the user's knowledge spans a set of topics satisfying the given attributes. We give an example.

Example 3.2 Assume that the user John-Doe knows topics with names "inverted index" at an expert (3) level, and "data compression" at a beginner (1) level, specified as

U-Knowledge(John-Doe) = {(TName = "inverted index", 3), (TName = "data compression", 1)}

Besides detail levels, we also keep the following history information for each topic source that the user has visited: Web addresses (URLs) of topic sources, their first/last visit dates and the number of times the source has been visited. The information on user's knowledge can be used during query processing, in order to reduce the size of the information returned to the user. We discuss query processing issues under user preferences and user knowledge in Chapter 4, along with Web query examples. In the absence of a user profile, the user is assumed to know nothing about any topic, i.e., the user's knowledge level about all topics is zero.

3.4 Creation and Maintenance of Expert Advice Repositories and User Profiles

In this section, we briefly discuss how the expert advice repositories and user profiles are constructed and maintained in order to demonstrate that metadata-based Web querying through our Web information space model is practically applicable.

3.4.1 Creation and Maintenance of Metadata Objects for a Subnet

With the fast increase in the amount of data on the Web, numerous tools for data extraction from the Web have been developed. A data extraction tool (e.g., wrapper) mines (meta)data from a given set of Web pages according to some mapping rules, and populates a (meta)data repository [48]. Such tools are generally based on several techniques such as machine learning, natural language processing, ontologies, etc. In [48], Laender et al. provide a survey for wrappers and they categorize them with respect to techniques employed during the data extraction.

The first step of creating metadata repositories is determining the topic and metalink types for the application domain. This is carried on by the domain experts either in a totally manual manner or by making use of thesauri or available ontologies. The second and more crucial step is discovering mapping rules to extract metadata from the actual Web resources, and this may involve techniques from machine learning, data mining, etc. (see [32, 83, 48] as examples). In this thesis, as we focus on metadata-based querying of subnets rather than the whole Web, the creation and maintenance of metadata repositories is an attainable task. Moreover, the advent of the XML over the Web can further facilitate such automated processes and allow constructing tools that will accurately and efficiently gather metadata for arbitrarily large subnets, with least possible human intervention.

For the performance evaluation experiments presented in Chapter 6, we created a metadata repository (namely, DBLP Bibliography metadata database) in a semi-automated manner by exploiting the DTD of the XML information resources. Essentially, we mapped topics and metalinks to the elements and attributes of DTD. Then, a Web robot traversed all the documents conforming to this DTD and populated the repository. To illustrate this semi-automated approach, consider the DTD given in Figure 3.1 for the DBLP Bibliography archive, which contains bibliographic information for computer science research papers.

```
<!ELEMENT dblp (article|inproceedings|proceedings|book|...)*>

<!ENTITY % field "author|editor|title|booktitle|year|
                  address|journal|URL">

<!ELEMENT article (%field;)*>
<!ELEMENT inproceedings (%field;)*>
<!ELEMENT proceedings (%field;)*>

<!ELEMENT author (#PCDATA)>
<!ELEMENT editor (#PCDATA)>
<!ELEMENT address (#PCDATA)>
...
```

Figure 3.1: A subset of DTD for XML documents in the DBLP Bibliography site.

According to this DTD, a `dblp` element may be an `article` (i.e., journal article), `proceedings` (i.e., conference proceeding), `inproceedings` (i.e., conference paper), a `book`, etc., and each of these elements may contain fields like `author`, `editor`, `title`, etc. Considering the DBLP DTD, a topic of type PaperName can be extracted by following the element tag path `dblp.inproceedings.title`, and getting the value between `<title>` and `</title>` tags, a topic of type AuthorName can be extracted from the path `dblp.inproceedings.author`, etc. Two topics $t1$ and $t2$ have *AuthoredBy* relationship between each other, if topic type of $t1$ is PaperName, and topic type of $t2$ is AuthorName, and both topics are extracted from the same `inproceedings` element instance. Similarly, topic source for a PaperName topic is extracted by following the path

```
<?xml version="1.0"?>
<!DOCTYPE dblp SYSTEM "dblp.dtd">
<dblp>
   <inproceedings key="...">
       <title>DMQL: A Data Mining Query Language for
              Relational Databases</title>
       <author>J. Han</author>
       <author>Y. Fu</author>
       <author>W. Wang</author>
       <author>K. Koperski</author>
       <author>O. Zaiane</author>
       <pages>...</pages>
       <booktitle>DMKD</booktitle>
       <year>1996</year>
       <crossref>conf/dmkd/</crossref>
       <url>http://www.informatik.uni-trier.de/
                 ~ley/db/conf/dmkd/HanFWKZ96.html</url>
   </inproceedings>
</dblp>
```

Figure 3.2: Example XML document

`dblp.inproceedings.URL`.

As an example, consider the XML document in Figure 3.2, which confirms to the DBLP DTD and contains bibliographic information about a conference paper. According to the above mapping rules, the metadata entities presented in Table 3.1 through Table 3.3 are extracted.

During the metadata creation, a domain expert also attaches importance scores to these metadata entities for providing more sophisticated querying facilities. Adding importance scores to topics, their sources and metalinks enriches the Web information space model by allowing query output ranking and size control. A query output is ranked with respect to metadata importance scores and limited to the highest-ranked topics/sources to save query processing time and improve the quality of query results as we discuss in the subsequent chapters.

Importance scores are attached to metadata entities in different forms:

Table 3.1: Topic instances for the XML Document in Figure 3.2

TId	**TName**	**TType**	**TDomain**	**TImp-Score**
T01	DMQL: A Data Mining Query Language for Relational Databases	Paper Name	Conference Paper	0.9
T02	DMKD-1996	Journal Conference and Year	Computer Science	0.9
T03	DMKD	Journal Conference Org	Computer Science	0.9
T04	1996	Publication Date	-	-
T05	J. Han	Author Name	Computer Science	1
T06	Y. Fu	Author Name	Computer Science	1
T07	W. Wang	Author Name	Computer Science	1
T08	K. Koperski	Author Name	Computer Science	1
T09	O. Zaiane	Author Name	Computer Science	1

Table 3.2: Topic source reference instances for the XML Document in Figure 3.2

TId	**URL**	**SImp-Score**
T01	http://www.informatik.uni-trier.de /∼ley/db/conf/dmkd/HanFWKZ96.html	1

Table 3.3: *AuthoredBy* metalink instances for the XML Document in Figure 3.2

MId	**Ant-Id**	**Cons-Id**	**MDomain**	**MImp-Score**
M01	{T05, T06, T07, T08, T09}	T01	Computer Science	1

- *Open form* [4]: For each metadata object in the repository, an expert manually assigns an importance score. As an example, we may have Imp(E.Topics, TName="DMQL: A Data Mining Query Language for Relational Databases", TType="PaperName", TDomain="Conference Paper") = 0.9 where Imp() denotes (a constant) importance score function and E.Topics denotes the topics table of the expert advice repository created by the expert E. This statement expresses that the domain expert assigns the importance score of 0.9 to the topic (paper) "DMQL: A Data Mining Query Language for Relational Databases" in the "Conference Paper" domain.

- *Closed form*: Each object's importance score is derived from a closed function. This approach, which we used during the creation of the DBLP Bibliography metadata database, is more practical to apply during automated or semi-automated metadata creation. For instance, the importance score for topics of type "PaperName" can be specified as a weighted function of citations received and the impact factor of the journal/conference in which the paper is published/presented. We express the importance score function in the closed form as Imp(E.Topics, TType= "PaperName") = f(no of citations, impact factor of the journal). In this case, the domain expert should also specify how to compute the function f() and determine each parameter in this function.

- *Semi-closed form*: Domain expert specifies a function to assign a score for a set of objects identified through regular expressions. Consider the function Imp(E.TopicSources, TName="*size of Web data*", TDomain="WWW", Last-Modified = (Now - 2years)) = No, where * denotes a wildcard character that matches any string. This function assigns the importance score "No" to all Web resources for any topic with topic name including the string "size of Web data" in the "WWW" domain and not updated in the last 2 years. So, these topic sources will never be included in query outputs unless they are updated.

More detailed discussion on creation of metadata repositories can be found in [5, 52], and [47] includes maintenance issues of metadata repositories when new

Web documents are included in the subnet. Matching of metadata objects from multiple metadata repositories is discussed in [68]. Once we create the metadata, we can store it in an object relational DBMS, and we can pose sophisticated queries for the underlying Web resources through the metadata database. In Chapter 4, we give examples of such queries.

3.4.2 Creation and Maintenance of User Profiles

In the Web information space model, our aim is to employ user profiles during metadata-based subnet querying to increase the quality of the results returned to the user. Thus, user profiles should be created and maintained by the querying application that makes use of our Web information space model. As user preferences allow each user to specify his/her preferences about experts and metadata objects, a Web querying application employing the Web information space model should allow its users to enter his/her own preferences by filling out a form, as shown in Figure 3.3. The user then, explicitly specifies which expert advice repositories he/she wants to use, as well as the other preferences (topic importance threshold, rejected resources, etc.).

As user knowledge maintains knowledge of users on topics in terms of detail levels and navigational history information for the users, it can be generated and updated from user click-stream data that is collected at the application level, i.e., search/query interface for a Web querying application based on our model. Assume that a user who login to such a Web querying application poses a query involving various metadata entities, and a list of required topic source URLs is returned. Then, as the user clicks some links in this list, the URL of the document that the user visits, the first and last visit dates, media type, and the visit frequency for the document are directly written to the user knowledge database. Besides, the detail level of each such topic source for the required topic in the query is retrieved from the expert advice repository and stored in the user knowledge.

Example 3.3 Assume that the user John Doe requires all sources for

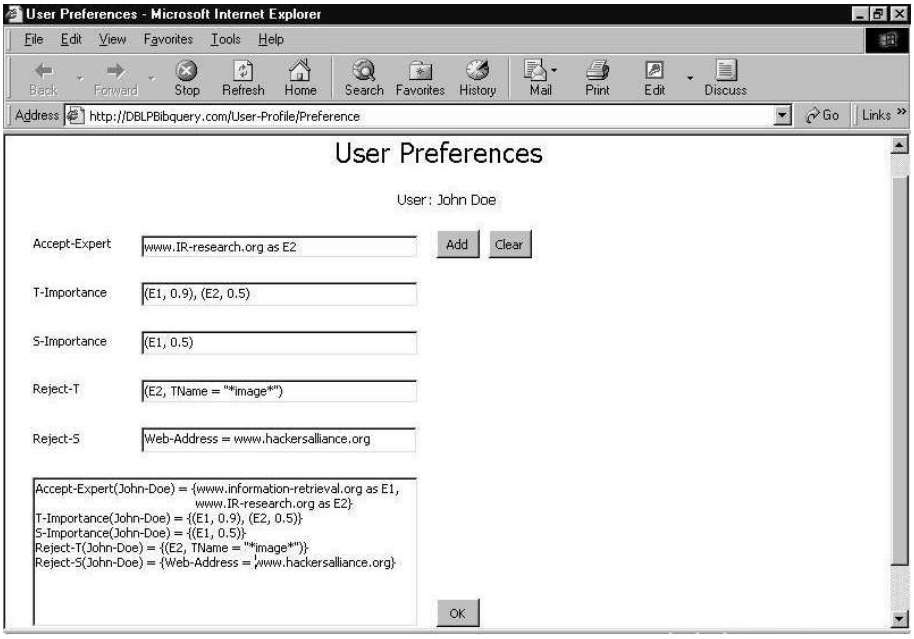

Figure 3.3: User preference specification form

the topic "inverted index", and the expert advice includes three sources www.IR-resources.org/inv-index.html (includes definition of inverted index), www.csindex.com/Baeza96.pdf (a paper containing introductory information about inverted indexes), www.DBLPBib.com/Moffat00.pdf (more advanced level paper about inverted indexes) for this topic with detail levels beginner (1), intermediate (2) and advanced (3), respectively. All three sources are returned to the user as the query response. Assume that the user knowledge formerly includes the entry U-Knowledge (John-Doe)={(TName="inverted index", 1)} and the user clicks to first and second sources. Then, his knowledge about this topic will be updated as "intermediate" and the entry becomes U-Knowledge (John-Doe)={(TName="inverted index", 2)}. Moreover, the list of visited URLs by the user John Doe is expanded with these two sources, along with their visit dates, media types, etc. The user knowledge database for this example is shown in Tables 3.4, and 3.5.

Information captured in user profile is employed for refining query results that are initially obtained by querying expert advice repositories. For instance, the user John Doe would specify that the sources for the topic "inverted index"

Table 3.4: Navigational history information for user John Doe

TName	**Detail Level**	**Web-Address**	**First Visit**	**Last Visit**	**Media Type**	**Freq**
inverted index	1	www.IR-resources.org/inv-index.html	1.2.03	2.12.03	text	11
inverted index	2	www.csindex.com/Baeza96.pdf	2.12.03	2.12.03	pdf	1

Table 3.5: Topic knowledge for user John Doe

TName	**TType**	**TDomain**	**TAuthor**	**Knowledge Level**
inverted index	Index Term	Information Retrieval	E	2

should be eliminated from the query output, if he has visited these resources in the last two weeks or the sources are at the "beginner" level. We discuss in the subsequent chapters the use of user profiles for query output refinement purposes in more detail.

Chapter 4

SQL Extensions and SVA Algebra

We model a subnet by employing our Web information space model, and store expert advices and user profiles in a (object) relational DBMS. For the purpose of personalized metadata-based subnet querying through our Web information space model, we extend the SQL with new clauses, specialized operators (e.g., text similarity based selection, text similarity based join, topic closure, etc.), and score management functionality. We also define Sideway Value generating Algebra (SVA) to support these SQL extensions.

4.1 SQL Extensions

We extend the basic "select" statement of SQL as below to query our Web information space model.

select <*Metadata Objects*>
using advice at <*URL of metadata database*> **as database** <*DB*>
[**using profile at** <*URL of user profile*> **as database** <*U*>]
from <*list of tables from DB and/or U*>

where <*Conditions*>
[**propagate importance as** <f> **function of** <*list of arguments*> |
topic closure importance computation as
<*FPath*> **function within a path and as**
<*FPathMerge*> **function among multiple paths**]
stop after <k> **most important**

- The clause **using advice at** <*URL of metadata database*> **as database** <*DB*> specifies the metadata database that is employed in the query. If more than one expert advice database is to be queried, a comma separated list of expert advice databases is specified in this clause.

- **using profile at** <*URL of user profile*> **as database** <U> clause specifies the user profile database that is used in the query. If the user specifies which expert advice that he/she wants to query in the user preferences, then **using advice at** clause may not be used in the query.

- **propagate importance as** <f> **function of** <*list of arguments*> specifies the formula for propagating importance scores of query input relations to the output relation. f is a monotonically decreasing function such as `min`, `product`, `average`, and `geometric average`, which always returns a value less than or equal to its input importance scores. *list of arguments* is a sublist of relations listed in the **from** clause of the query. Only the importance scores for the relations specified in the **propagate importance** clause are employed during importance score computation of output tuples.

- Queries involving a topic closure operator should include topic closure importance computation clause

  topic closure importance computation as
  <*FPath*> **function within a path and as**
  <*FPathMerge*> **function among multiple paths**

  which specifies how to compute the derived importance scores of topics encountered during topic closures. The functions *FPath* and *FPathMerge*

are functions like `product`, `max`, `min`, etc., and we describe what kind of functions can be employed in the topic closure importance propagation clause in Section 4.2.4, where we present the topic closure operator. If an extended SQL query includes a topic closure operator, the query must have **topic closure importance computation** clause, otherwise **propagate importance** clause is used.

- The query stopping clause **stop after** $<k>$ **most important** specifies the ranking threshold such that the query returns at most k objects having the highest derived importance scores as output.

We give examples of extended SQL queries in the next section where we describe the Sideway Value generating Algebra operators.

4.2 Sideway Value Generating Algebra

In the Web information space model, we attach importance scores to topic, metalink, and topic source reference objects. We refer to these values as *sideway values*[3] which are used for ranking query outputs and limiting output size during query evaluation. We extend SQL with new algebraic operators supporting score (sideway value) management functionality, and that's why we called the underlying algebra Sideway Value generating Algebra (SVA). The SVA operators modify and propagate sideway values of base relations, and employ these values for efficient query processing.

As our aim is to provide efficient querying of Web-based information resources, our SQL extensions also allow approximate text similarity based querying, which is supported by majority of today's Web search engines. We define text similarity based selection (i.e., similarity based SVA selection), two types of text similarity based join (i.e., similarity based SVA join, and similarity based SVA directional join) operators to perform similarity based querying. In addition to these, we

[3]We use the terms "importance score" and "sideway value" interchangeably in this thesis.

define a new operator, SVA topic closure, to allow query posers to formulate useful queries which may not be performed by typical keyword matching based querying system.

We describe SQL extensions, SVA operators, and score management through example queries in the subsequent sections. In the logical query tree examples discussed next, we use the following notation: Operators with superscript * are SVA operators. Operators without superscript * are normal relational algebra operators. A unary relational algebra operator without * in its superscript simply carries (into its output tuples) the sideway values of its operand relation. A binary relational algebra operator without a superscript * may carry (into its output tuples) sideway values of either its left hand side relation or its right hand side relation, indicated by superscript L or R, respectively.

4.2.1 Similarity Based SVA Selection Operator

Our text similarity based SVA selection operator is represented and defined as follows.

Notation: $\sigma^*_{sC,f_{out},k}(R)$

Definition: The selection operator σ^*_s takes as input a relation R with importance propagation function f_{in}, a text similarity based selection condition C, an output importance propagation function f_{out} for the output tuples, and a positive integer k as the ranking threshold. The operator σ^*_s returns, in decreasing order of output importance scores, k output tuples that satisfy the selection condition C.

The text similarity based SVA selection operator can be employed during text similarity based search over the Web information space model, which is a required search facility, as the majority of Web based information resources consist of text, and experts assign arbitrary names to topics. This allows query poser to retrieve topics even if he/she does not know the exact topic name.

Example 4.1 Using the advice at www.DBLPandAnthology.com/advice, find

the names and URLs of 10 highest topic importance ranked journals and conferences having names similar to the string "Web data management". Employ a product based importance propagation function.

select T.TName, S.URL
using advice at www.DBLPandAnthology.com/advice **as database** DB
from DB.Topics T, DB.TSRef S
where T.TType="JournalConferenceOrg" **and**
T.TName ≅ "Web data management" **and**
T.TId **in** S.TId
propagate importance as product **function of** T
stop after 10 **most important**

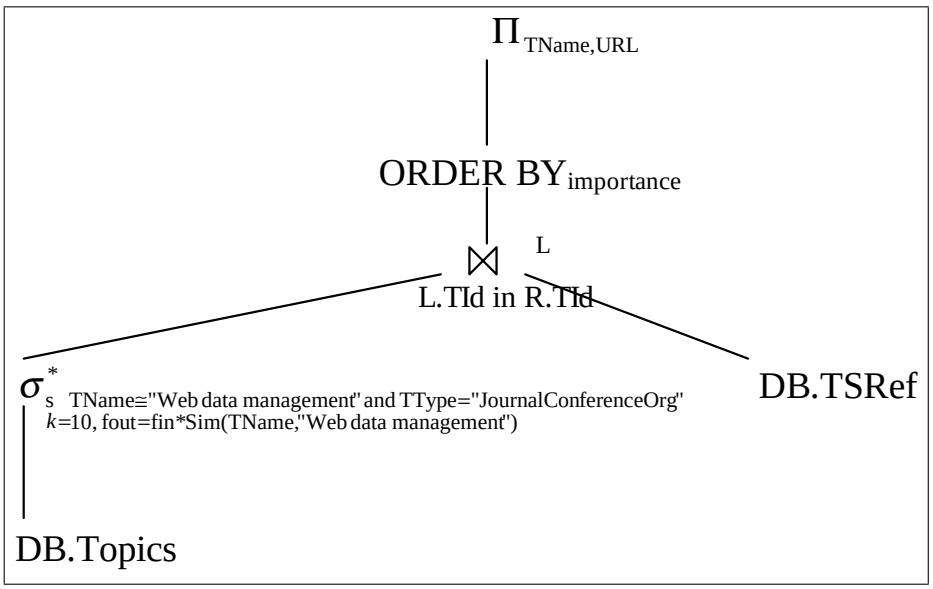

Figure 4.1: Logical query tree for Example 4.1.

The logical query tree of Example 4.1 is shown in Figure 4.1. This query chooses journal/conference names and their URLs on the basis of the "derived" importance values of JournalConferenceOrg topics as described in the "propagate importance" clause. The clause T.TName ≅ "Web data management" states that the selection condition is similarity based, and k topics having the highest derived importance scores and having topic names similar to the string "Web

data management" are selected. The function Sim() in Figure 4.1 computes the text similarity (according to tf-idf similarity measure, see Chapter 5) of two strings, and returns a value in the range [0, 1]. Here, Sim() is used to modify the importance scores of output tuples according to their TName similarity to the string "Web data management". Due to the "propagate importance" clause, product function is employed as f_{out} function, and the value of k equals 10 because of the "stop after" clause. The URLs for the selected journal/conference topics can be easily obtained by joining the output tuples of the SVA selection operator with the T(opic)S(ource)Ref(erence) table, and the join operator in Figure 4.1, passes the importance scores of its left hand side input relation to its output.

Algorithm: Text Similarity Based SVA Selection

Input: Relation R, an inverted index I on relation R,
selection condition C, an importance propagation function f_{out},
an integer k.
Output: Selected tuples with respect to importance scores.

begin
for each term t in text similarity based condition (Q) in C **do**
Search t from the inverted index I.
if it exists in the inverted index
for all tuples p in the inverted list entry for term t **do**
if p also satisfies other selection conditions (if any) **then**
$\diamond$ Accumulate similarity $Sim(p.TName, Q)$;
$\diamond$ $p.score = f_{out}(p.score) * Sim(p.TName, Q)$;
$\diamond$ Store p and $p.score$ in list S.
end if
end for
end for
Return k tuples from S having the highest modified importance scores.
end

Figure 4.2: Similarity based SVA selection algorithm

The query processing algorithm for the SVA selection operator is presented in Figure 4.2. A naive SVA selection algorithm scans the input relation once, and for each tuple in the relation, the algorithm checks whether the tuple satisfies

the selection conditions or not. If the tuple satisfies the selection conditions, the algorithm computes the modified importance score for the tuple, and at the end, outputs k tuples having the highest modified importance score. For the naive algorithm, we need to have topic name vectors for each tuple to compute the similarity of topic name of the tuple with the string constant specified in the text similarity based selection condition. In the algorithm we present in Figure 4.2, we employ an inverted index (for topic name attribute) over the input relation, and we do not pass over the whole relation. We make similarity computations, and check selection conditions only for tuples which are guaranteed to have a similarity value greater than zero to the string constant given in the text similarity based selection condition. We discuss the effects of employing inverted indexes on the performance of query processing of similarity based SVA directional join operator in Chapter 5. The results obtained for the similarity based SVA directional join operator are also applicable for the SVA selection operator.

4.2.2 Similarity Based SVA Join Operator

Another SVA operator is the text similarity based SVA join operator.

Notation: $(L) \bowtie^*_s {}_{A \cong B, f_{out}, k} (R)$

Definition: The text similarity based join operator $\bowtie^*_s$ takes as input two relations L and R with sideway value functions fl_{in} and fr_{in} respectively, a text similarity based join condition on attributes A and B of relations L and R, respectively, a sideway value propagation function f_{out} for the output tuples, and an output size threshold k. The join operator then joins tuples of L and R if their attributes A and B are similar to each other, computes importance scores of output tuples as specified by f_{out}, and returns k joined tuples having the highest importance scores.

Text similarity based SVA join operator is useful when topics from different expert advices are joined, as each expert may assign different names to same topics. The join operator is also employed for retrieving similar topic pairs from

the same expert advice.

Example 4.2 Using the advices at www.DBLP.com/advice (E1) and www.SIGMOD.org/advice (E2), for the index terms advised by expert E1, find the most similar index terms from E2, and return top 20 index term pairs with the highest derived importance scores. Employ a geometric average based importance propagation function.

select T1.TName, T2.TName
using advice at www.DBLP.com/advice **as database** DB1,
www.SIGMOD.org/advice **as database** DB2
from DB1.Topics T1, DB2.Topics T2
where T1.TType="IndexTerm" **and**
T2.TType="IndexTerm" **and**
T1.TName ≅ T2.TName
propagate importance as gmtrc-avg **function of** T1, T2
stop after 20 **most important**

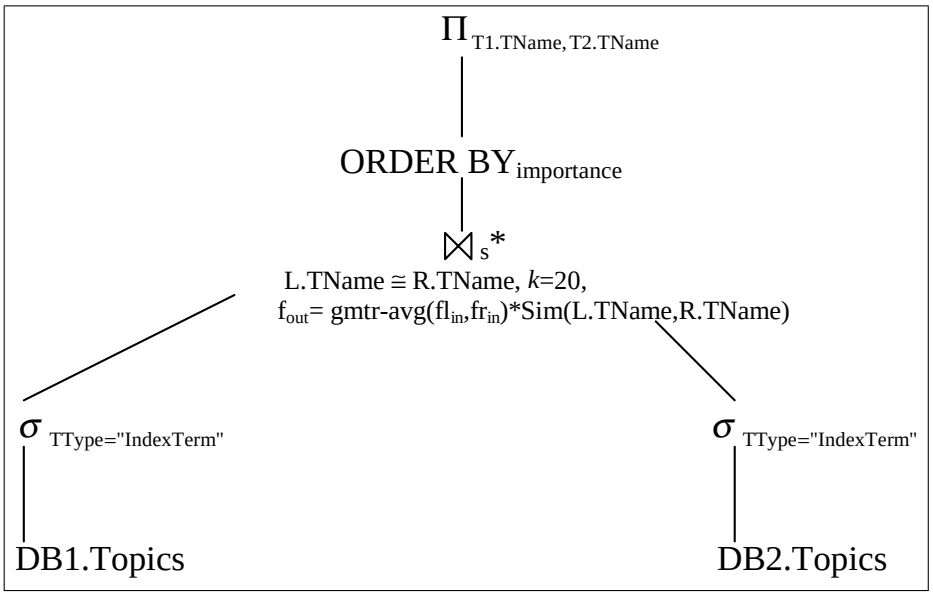

Figure 4.3: Logical query tree for Example 4.2.

In this query, text similarity based join operation is performed between the "IndexTerm" topics from the two expert advice databases. Topics of type index

term are joined if their topic names are similar to each other. The importance score of the joined tuples are computed as the geometric average of importance scores of input relations, and this value is revised by the similarity value of the index terms (their topic names). Then, the join operator returns 20 joined tuples having the highest modified importance score. The logical query tree for this query is presented in Figure 4.3. In the query tree, relational algebra selection operators select index term type topics and pass the importance score of their input relations without any modification. The text similarity based SVA join operator then, modifies the importance score according to the function provided in the "propagate importance" clause.

Text similarity based SVA join algorithms and their performance evaluation experiments are presented in [69, 70]. The algorithms that have been proposed in [69, 70] are nested loop based join algorithms employing an inverted index on one of the input relations to decrease the number of similarity computations performed.

4.2.3 Similarity Based SVA Directional Join Operator

The other text similarity based SVA operator is directional join operator, which is different from the similarity based SVA join operator as described below.

Notation: $(L) \bowtie^{*}_{dir\ A \cong B, f_{out}, k} (R)$

Definition: The text similarity based SVA directional join operator $\bowtie^{*}_{dir}$ takes as input two relations L and R with sideway value functions fl_{in} and fr_{in} respectively, a text similarity based join condition on attributes A and B of relations L and R, respectively, a sideway value propagation function f_{out} for the output tuples, and an integer k. The join operator then joins each tuple l of relation L with at most k tuples from relation R such that the derived importance score for tuple r of relation R (i.e., $f_{out}(fl_{in}, fr_{in})$* Sim($l.A$,$r.B$)) is among the top k highest derived importance score for the joined tuple $l.r$.

Text similarity based SVA directional join operator may be employed in integration and querying of multiple expert advices to facilitate metadata-based subnet querying.

Example 4.3 Using the advice at www.DBLPandAnthology.com/advice, find 5 papers having the most similar titles to each paper whose title includes the string "association rule mining" and was written by "J. D. Ullman". Employ a product based importance propagation function.

select T1.TName, S1.URL, T2.TName, S2.URL
using advice at www.DBLPandAnthology.com/advice **as database** DB
from DB.Topics T1, DB.Topics T2, DB.AuthoredBy M,
DB.Topics T3, DB.TSRef S1, DB.TSRef S2
where T1.TType="PaperName" **and** T2.TType="PaperName" **and**
T1.TName $\cong_{(dir,k=5)}$ T2.TName **and**
T3.TType="AuthorName" **and** T3.TName="J. D. Ullman" **and**
T3.TId **in** M.AntId **and** T1.TId=M.Cons-Id **and**
T1.TName **like** "*association rule mining*" **and**
T1.TId **in** S1.TId **and** T2.TId **in** S2.TId
propagate importance as product **function of** T1, T2

The logical query tree of Example 4.3 is shown in Figure 4.4. The clause "T1.TName $\cong_{(dir,k=5)}$ T2.TName" represents text similarity based SVA directional join operator, which joins each T1 topic with at most 5 T2 topics having the highest derived importance score according to the function specified in the "propagate importance" clause. Cons-Id and Ant-Id are consequent and antecedent topic attributes of the *AuthoredBy* metalink entity. We assume that *AuthoredBy* is a metalink type that specifies the relationship between a research paper and its set of authors. The signature of the metalink type is *AuthoredBy*: Set of AuthorName → PaperName. We also assume that, topic ids rather than topic names for the topics of types AuthorName and PaperName are stored in the *AuthoredBy* metalink table. In this query, we do not need "stop after" clause as the ranking threshold is specified in the SVA directional join statement.

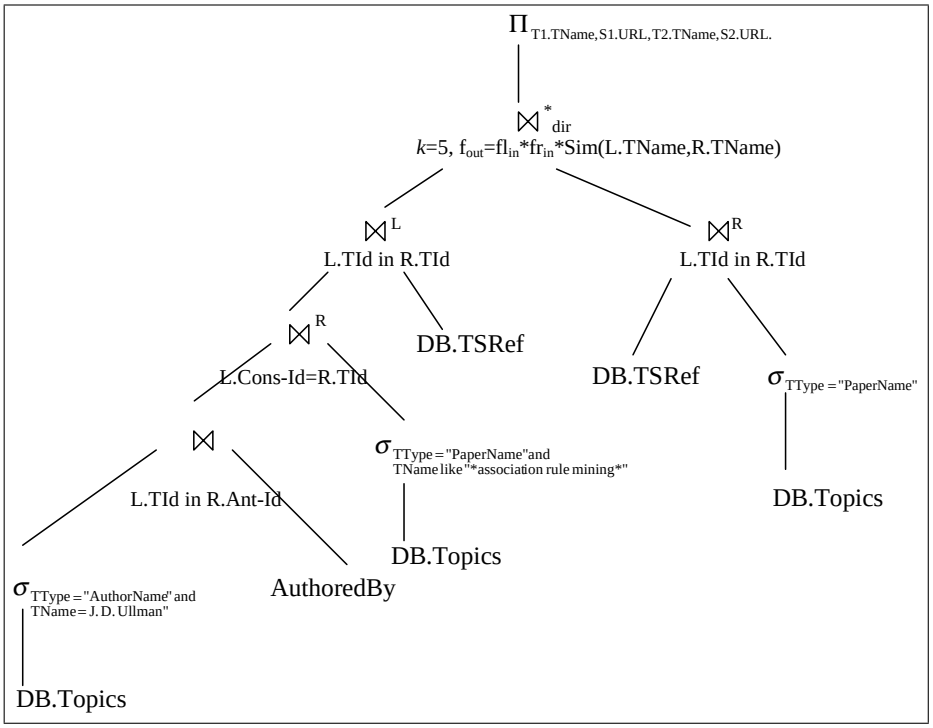

Figure 4.4: Logical query tree for Example 4.3.

Example 4.4 Let's assume that an editor of a journal wants to find 5 referees for each paper submitted to the journal. Also assume that, all submitted papers (pdf or ps files), along with their titles, keywords, author names, name and address of the corresponding authors are stored in a database. And additionally, an expert E provides metadata about referees such as their names, affiliations, and research areas. The editor can easily find the top 5 most suitable referees for each paper by using the text similarity based SVA directional join operation on the relation that contains information about submitted papers and the relation maintained for referees. The joining attributes of this join operation are the keyword attribute of papers and the research interest attribute of referees.

The text similarity based SVA directional join operator produces different output for $(L) \bowtie^*_{dir\ A \cong B, f_{out}, k} (R)$, and $(R) \bowtie^*_{dir\ B \cong A, f_{out}, k} (L)$ operations. The order of the operands is important, that is why we called this operator "directional". As an example, let L and R be relations on papers and referees, respectively, as discussed in Example 4.4. Then $(L) \bowtie^*_{dir\ A \cong B, f_{out}, k} (R)$ finds the top-5 most suitable referees for each submitted paper, whereas $(R) \bowtie^*_{dir\ B \cong A, f_{out}, k} (L)$ finds

the top-5 most suitable papers for each referee.

We discuss query processing algorithms for the text similarity based SVA directional join operator in Chapter 5, and give results for performance evaluation experiments.

4.2.4 SVA Topic Closure Operator

SVA topic closure is a recursive closure operator that takes into account the rankings of its input tuples.

Notation: $TClosure^{*}_{Topics,MetalinkType,FPath,FPathMerge,k}(X)$

Definition: The SVA topic closure operator computes the topic closure X^{+} of a set X of topics with respect to a metalink type (and, thus, with respect to the set of axioms characterizing the metalink type). The operator takes as input three relations, namely, the relation X, the relation $Topics$ containing all topic instances, and the relation $MetalinkType$ containing all instances of given metalink type; $FPath$ function that specifies how to compute the importance values of newly reached topics with respect to a single path, the function $FPathMerge$ that specifies how to merge the importance values obtained for different paths, and the ranking threshold k. It then computes the closure where each new topic in the closure is represented as an output tuple, and has a derived importance score which is among the top k highest derived importance scores.

Example 4.5 Using the advice at www.DBLPandAnthology.com/advice, find the titles and URLs of 10 highest importance-valued papers that are prerequisites (recursively) of the paper "DMQL: A Data Mining Query Language for Relational Databases".

select T2.TName, S2.URL
using advice at www.DBLPandAnthology.com/advice **as database** DB
from DB.Topics T, DB.Topics T1, DB.Topics T2,
DB.PrerequisitePapers M, DB.TSRef S2

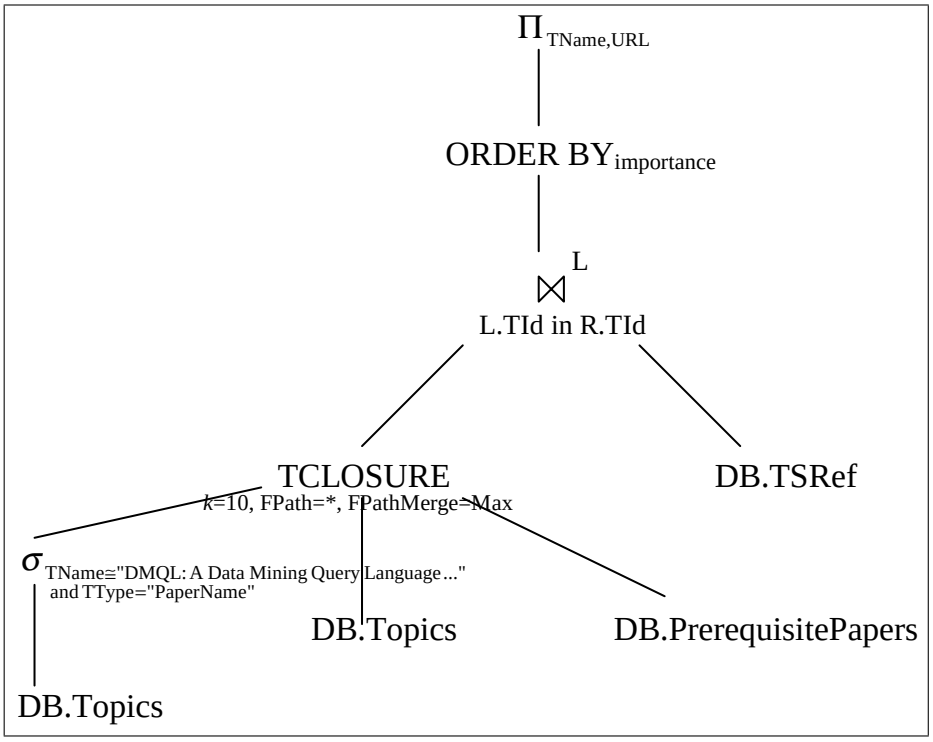

Figure 4.5: Logical query tree for Example 4.5

where T1.TName=“DMQL: A Data Mining Query Language
for Relational Databases” **and**
T1.TType=“PaperName” **and**
T2.TId **in** *PrerequisitePapers**(T1.TId, T, M) **and**
T2.TId **in** S2.TId
topic closure importance computation as
product **function within a path and as**
max **function among multiple paths**
stop after 10 **most important**

In the above query, prerequisites of the paper “DMQL: A Data Mining Query Language for Relational Databases” are located recursively by following the metalink instances of type *PrerequisitePapers*. The statement “T2.TId **in** *PrerequisitePapers**(T1.TId, T, M)” is topic closure clause and the function types for *FPath* and *FPathMerge* are specified in the “topic closure importance computation” clause. In this query, the closure operator first finds all instances of

PrerequisitePapers metalink type such that the paper "DMQL: A Data Mining Query Language for Relational Databases" is antecedent topic. Then, it gets all consequent topics, and revises their importance scores by using the *FPath* and *FPathMerge* functions. The derived importance score of a consequent topic is computed as $(ant\text{-}topic.score) * (m.score) * (cons\text{-}topic.score)$ as the *FPath* function is `product`. m is the metalink instance from which the consequent topic is reached, *ant-topic* is the antecedent topic, and *cons-topic* is the consequent topic of the metalink instance m. The reached topics and their derived importance scores are stored in a list, and the prerequisite topics for the newly reached topics are found by following the *PrerequisitePapers* metalink paths recursively. This process is repeated for all the topics in the list, until no new topic with higher revised importance score is reached. And, the topics having the $k(=10)$ highest derived importance scores are returned as output. If a topic is reached by following more than one *PrerequisitePapers* metalink paths, then the maximum score among the paths is taken as the derived importance score for the topic, as the *FPathMerge* function is `max` for this query. The logical query tree for this example is given in Figure 4.5.

If the query poser wants to see top 5 papers having titles that are most similar to the title "DMQL: A Data Mining Query Language for Relational Databases", and their top 10 most topic important prerequisites, then he/she will need to run the below queries.

select T.TId, T.TName
using advice at www.DBLPandAnthology.com/advice **as database** DB
from DB.Topics T
where T.TType="PaperName" **and**
T.TName ≅ "DMQL: A Data Mining Query Language
for Relational Databases"
propagate importance as product **function of** T
stop after 5 **most important**

select T1.TName, T2.TName, S2.URL

using advice at www.DBLPandAnthology.com/advice **as database** DB
from DB.Topics T, T1, DB.Topics T2,
DB.PrerequisitePapers M, DB.TSRef S2
where T2.TId **in** *PrerequisitePapers**(T1.TId, T, M) **and**
T2.TId **in** S2.TId
topic closure importance computation as
product **function within a path and as**
max **function among multiple paths**
stop after 10 **most important**

In the first query, 5 similar and having highest derived importance score valued papers are found, and the query poser stores the output in a table named T1. Then, T1 is included in the second query to find the prerequisite papers.

In the "topic closure importance computation" clause, the function $FPath$ should be a monotonically decreasing function. We define the $FPath$ function as follows: let $FPath$ takes a set of reals in the range $[0, 1]$ and returns a real in $[0, 1]$, and S be a nonempty set of reals in $[0, 1]$ and v be a real in $[0, 1]$. Then, $FPath(S \bigcup \{v\}) \leq FPath(S)$. This property of $FPath$ guarantees that the search for topics over a metalink path always comes to an end. That is, a topic obtained over a path that includes topic t (and, thus, is reached after t is reached) always has a propagated importance value lower than the propagated importance value of t. To guarantee that, during topic closure computations, the search for topics over multiple and possibly merging paths comes to an end, the $FPathMerge$ function has the following property: Assume that the input of $FPathMerge$ is the set $S = \{v_1, ..., v_n\}$ where v_i is a real in the range $[0, 1]$ for $1 \leq i \leq n$. Then, $FPathMerge(S) \leq Max(S)$.

The query processing algorithm that we employ in this thesis for the SVA topic closure operator is presented in Figure 4.6. More general topic closure algorithms that computes closure with respect to a regular expression of multiple metalink types are presented in [69, 70].

Algorithm: SVA Topic Closure (for a topic t)

Input: Topic t for which topic closure is computed,
relation *Topics* which includes all topics in the expert advice,
relation M including all instances of metalinks of a given type,
FPath and *FPathMerge* functions, and an integer k.
Output: k topics having the highest derived importance scores and
logically implied from topic t and metalink instances in M.

begin
$Closure = \{\}$; $Candidate = \{\}$.
Get the TId and importance score of t (i.e., $t.score$).
$Closure = Closure \bigcup \{t.TId\}$
for each topic $t_1 \in Closure$ **do**
i. $Closure = Closure - \{t_1\}$; $Candidate = Candidate \bigcup \{t_1\}$.
ii. $m \in M$; $t_1.TId = m.Ant\text{-}Id$; $t_2 = m.Cons\text{-}Id$;
iii. **for** each topic t_2 **do**
◇ Compute derived importance of t_2 as
$t_2.score' = FPath(t_1.score, t_2.score, m.score)$.
if t_2 is not in the $Candidate$ set **then**
if t_2 is not in the $Closure$ set
∘ $Closure = Closure \bigcup \{t_2\}$;
∘ $t_2.score = t_2.score'$;
else
∘ $t_2.score'' = FPathMerge(t_2.score', t_2.score)$;
end if
end for
iv. Sort the topics in $Closure$ with respect to their
derived importance scores in descending order.
v. if $|Candidate| \geq k$ and minimum score in the $Candidate$
set is greater than the maximum score in the $Closure$ set **then**
Exit the loop.
end for
Return k topics having the highest derived importance scores from
the $Candidate$ set.
end

Figure 4.6: SVA topic closure algorithm

4.2.5 Other SVA Operators

For each relational algebra operator, there is an SVA counterpart extended with an output sideway value function f_{out} and the output ranking threshold k which is an integer value. The SVA operator processes its input relation in the same way that its relational algebra counterpart does, and also the SVA operator modifies the importance score of its input relation with the function f_{out} and gives the k tuples having the highest modified importance score as output.

Example 4.6 Using the advice at www.DBLPandAnthology.com/advice, find the names and URLs of 10 highest topic importance ranked journals and conferences whose scope involves the term "query processing". Employ a product based importance propagation function that uses all involved importance values.

select T2.TName, S.URL
using advice at www.DBLPandAnthology.com/advice **as database** DB
from DB.Topics T1, DB.Topics T2, DB.TSRef S, DB.HasScope M
where T1.TType="IndexTerm" **and** T1.TId **in** M.AntId **and**
T1.TName="query processing" **and**
T2.TType="JournalConferenceOrg" **and**
T2.TId=M.Cons-Id **and** T2.TId **in** S.TId
propagate importance as product **function of** T2, M
stop after 10 **most important**

The query tree for this example is given in Figure 4.7. In the figure, the SVA join operator modifies the importance score of its input relations, however, the join condition is not similarity based (i.e., exact).

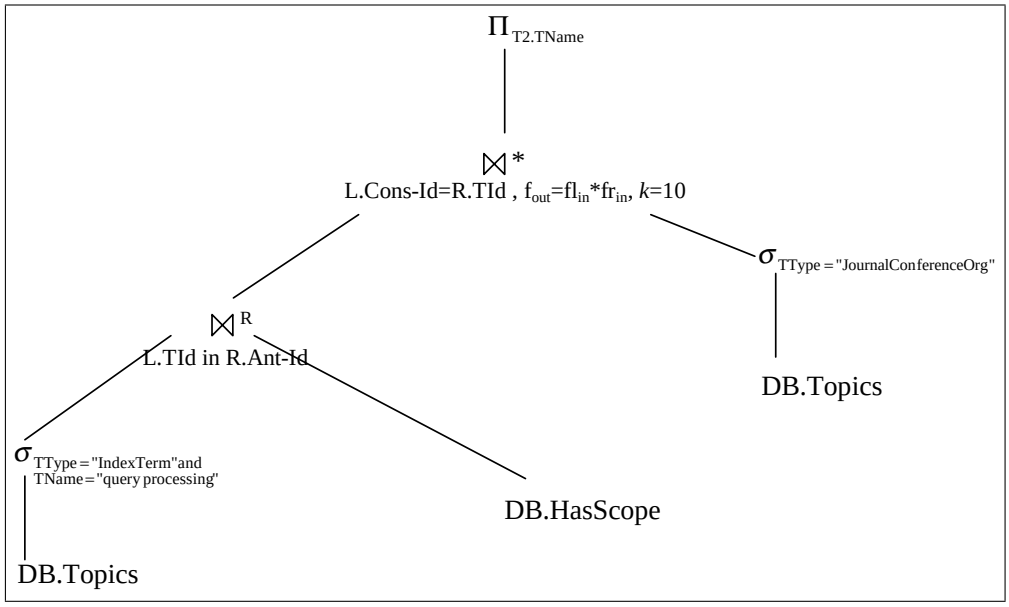

Figure 4.7: Logical query tree for Example 4.6.

4.3 Extended SQL Queries with User Profiles

Our SQL extensions allow user profiles to be included into the queries to facilitate personalized metadata-based subnet querying. User profiles include user preferences about experts, topics, sources, and metalinks, as well as user knowledge about topics. An extended SQL query employing user profiles can be processed by performing relational algebra selection operators over topics, metalinks, and topic source reference tables, as we show in the below example.

Example 4.7 Using the profile at www.DBLPandAnthology.com/profile/John-Doe, find the names and URLs of 10 highest topic importance ranked papers having the index term "query processing". Employ a product based importance propagation function that uses importance values of papers and metalink instances.

Let's assume that the user profile employed in this query specifies the below preferences, in which the user wants to employ only one expert advice, see paper names including the string "relational databases", does not want any topic with topic name including the string "parallel", and include only the topics and sources

having an importance value of at least 0.5 in the query output.

Accept-Expert(John-Doe) = {www.DBLPandAnthology.com/advice as E1}
T-Importance(John-Doe) = {(E1, 0.5)}
S-Importance(John-Doe) = {(E1, 0.5)}
Accept-T(John-Doe) = {(E1, TName="*relational database*",
TType="PaperName")}
Reject-T(John-Doe) = {(E1, TName= "*parallel*",
TType="PaperName")}

The extended SQL query is formulated as follows:

select T2.TName, S.URL
using profile at www.DBLPandAnthology.com/profile/John-Doe **as database** U
from U.E1.Topics T1, U.E1.Topics T2, U.E1.TSRef S, U.E1.IndexedBy M
where T1.TType="IndexTerm" **and** T1.TId **in** M.Ant-Id **and**
T1.TName="query processing" **and**
T2.TType="PaperName" **and**
T2.TId=M.Cons-Id **and** T2.TId **in** S.TId
propagate importance as product **function of** T2, M
stop after 10 **most important**

The query tree for this example is presented in Figure 4.8. In the query tree, user preferences are satisfied through extra selection conditions in addition to the selection conditions that are specified explicitly in the query.

Further examples of extended SQL queries are presented in Chapter 6 and Appendix A.

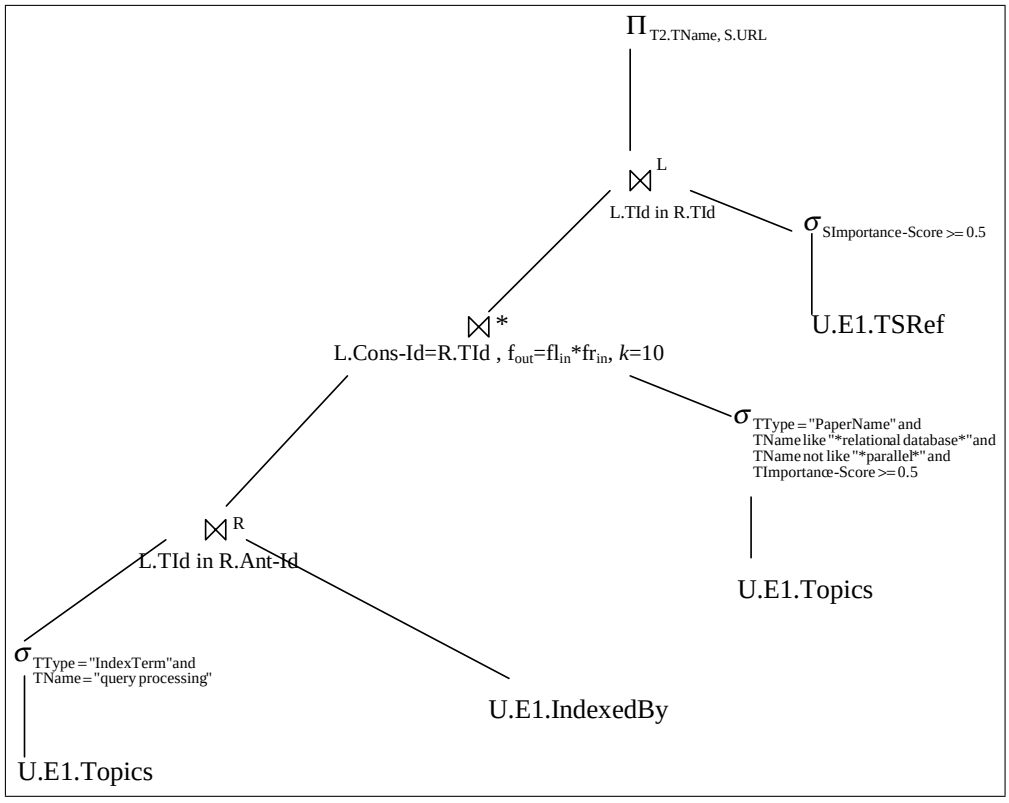

Figure 4.8: Logical query tree for Example 4.7.

Chapter 5

Similarity Based SVA Directional Join

Various methods have been proposed for the efficient implementation of the join operator which is known to be one of the most expensive operations in relational databases. Yang et al. and Mishra et al. provide survey of (equi)join techniques for relational databases [60, 85]. While a large amount of work has been devoted to processing of equijoins, only a few papers have appeared in the literature regarding the text similarity based join operator [25, 59]. The text similarity based join operator, as we describe in Chapter 4, joins two relations if their join attributes, which consist of pure text, are similar to each other. The similarity between join attributes is determined by well-known techniques such as tf-idf weighting scheme [75] and cosine similarity measure from the Information Retrieval (IR) domain, for which we give a brief explanation in Section 5.1. The text similarity based join operator has several application domains: Cohen used this operator for the integration of data from distributed, heterogeneous databases that lack common formal object identifiers [25]. For instance, in two Web databases listing research institutions, to determine whether the two names "AT&T Labs" and "AT&T Research Labs" denote the same institution or not, text similarity based join operator may be employed.

Meng et al. used the text similarity based join operator to query a multidatabase system that contains local systems managing both structured data (e.g., relational database) and unstructured data (e.g., text) [59]. Assume that we have two global relations: applicants containing information about job applicants and their resumes, and positions including the description of each job; then the text similarity based join operator is used to answer queries like "for each position, find k applicants whose resumes are most similar to the position's description".

We use the text similarity based join operator for the integration and querying of metadata from multiple resources to facilitate metadata-based Web querying. We give examples for such queries in Chapter 4. Besides, we define two types of text similarity based join operator, namely the text similarity based SVA join operator and the text similarity based SVA directional join operator in Chapter 4. Both types of operators take two relations L and R, and an integer k as input. The former join operator joins k similar tuple pairs (according to the similarity of their textual join attributes) having the highest derived importance scores from L and R, while the latter one joins each tuple in relation L with k similar tuples having the highest derived importance scores from relation R. Thus, text similarity based SVA join operator produces output relation of size at most k tuples, while the maximum number of tuples in the output relation of similarity based SVA directional join operator is *number of tuples in relation* L * k. Text similarity based SVA join operator, text similarity based SVA join algorithms, and performance evaluations of these algorithms are presented in our recent work [69]. In this thesis, we only focus on the processing of the text similarity based SVA directional join operator and provide efficient algorithms for this operator.

This chapter is organized as follows. In the next section, we define the similarity measure employed in similarity comparisons of tuple attributes. A brief summary of the previously proposed algorithms are presented in Section 5.2. Our new join algorithm, and early termination heuristics applied to our algorithm are explained in Section 5.3. After that, we experimentally evaluate and compare all the algorithms in terms of the number of tuple comparisons, the number of disk accesses made, and the amount of main memory required in Section 5.4. Finally,

we conclude our discussion in Section 5.5.

5.1 The Similarity Measure

The similarity measure used in all similarity based SVA operators is the cosine similarity measure with tf-idf weighting scheme [75]. The previous work on similarity based text join also employed the same similarity measure [25, 59]. In this measure, each document (join attribute in the text similarity based SVA directional join operator) is represented as a vector consisting of n components, n being the number of distinct terms in the document collection. Each component v_i of a vector v for a document d gives the weight of the term i for that document. Weight of a term for a particular document is computed according to $tf * idf$ value, where tf stands for *term frequency* meaning that the number of occurrences of term i within the document; *idf* is *inverse document frequency* and it gives more weight to scarce terms in the collection. The $tf * idf$ rule states that a term that appears in many documents should not be regarded as being more important than a term that appears in a few documents. Also, a document with many occurrences of a term should not be regarded as being less important than a document that has just a few [82]. The similarity measure is the cosine of the angle between the two document vectors such that the larger the cosine, the greater the similarity. For the text similarity based SVA directional join algorithm, we use the following formulas to give term weights for each tuple and measure the similarity between textual attributes:

$$w_{l,t} = \log_e(1 + N/f_t) \tag{5.1}$$

$$w_{r,t} = 1 + \log_e(f_{r,t}) \tag{5.2}$$

$$W_l = \sqrt{\sum_{t \in l} w_{l,t}^2} \tag{5.3}$$

$$W_r = \sqrt{\sum_{t=1}^{n} w_{r,t}^2} \tag{5.4}$$

in which N denotes the total number of tuples in relation R, f_t is the total number of tuples in R that contain term t. $w_{l,t}$ is the weight of term t for tuple l in relation L. $w_{r,t}$ denotes the weight of term t for tuple r in relation R. $f_{r,t}$ is the frequency of term t in tuple r. n is the total number of distinct terms in the collection (relation R). W_l and W_r are the normalization factor for tuple l and tuple r, respectively. The similarity between tuples l and r is calculated by using the following formula:

$$cos\_sim(l, r) = \frac{1}{W_l \cdot W_r} \sum_{t \in (l \cap r)} (w_{l,t} \cdot w_{r,t}) \tag{5.5}$$

As an example, assume that the join attributes of tuples l and r consist of the strings "A Query Language for Relational Database Systems", and "Query Processing for Relational Databases", respectively. Also assume that, the relation R has 132,000 tuples, and 8,563 of them contain the term "database". Then, the weight of the term "database" for tuple l is computed as $log_e(1 + 132000/8563)$ which equals 2.74. The weights of all terms in tuple l are computed similarly. For tuple r, the weight of the term "database" is calculated as $1 + log_e(1)$ since the frequency of the term is 1. After all the term weights are calculated for tuples l and r, the similarity between them is computed by using the cos_sim formula presented above.

In the above weighting scheme, to calculate weights for each tuple in relation L, we use the statistical data for relation R. Because, in the SVA directional join operator, our aim is, for each tuple in L, to find most similar (and having the highest derived importance score) tuples from relation R. And, the weighting scheme we describe above is employed in search engines to find k most similar documents for a given user query [82]. Thus, each tuple l in relation L is considered as a user query, and the relation R as the document collection of search engines.

For the text similarity based SVA directional join operation, we do not need to make normalization with respect to tuple l, since for tuple l we want to find the k tuples having the highest derived importance score tuples from R, and

dividing similarity values of R tuples to the same real value W_l does not affect the relative order of the similarity values. Thus, we can eliminate some of the expensive operations (multiplication, division, square-root) that are employed in the similarity computation.

Other measures such as Hamming distance, and longest common subsequence (LCS) for determining the similarity between short strings have also been developed. We prefer tf-idf weighting scheme and cosine similarity measure because as it is shown in [25, 36] they give quite good matches even for short strings. Also, the tf-idf weighting allows the use of inverted indices, which enables us to employ some early termination heuristics from the IR domain during the similarity comparisons of tuples.

5.2 Text Similarity Based Join Algorithms

The only algorithms that have appeared in the literature for the similarity based text join operator were developed by Meng et al. [59] and Cohen [25]. Meng et al. presented three algorithms namely HHNL, HVNL, and VVM for the text join operator. The HHNL (Horizontal-Horizontal Nested Loops) algorithm, as its name implies, is a nested loops based join algorithm, in which each tuple l in relation L is compared with every tuple in relation R, and k most similar tuples from R are joined with tuple l. This is the straightforward way for evaluating the join operation. In [59], the input relations L and R are read from disk. After reading X tuples from L into the main memory, the tuples in R are scanned; and while a tuple in R is in the memory, the similarity between this tuple and every tuple in L that is currently in the memory is computed. For each tuple l in L, the algorithm keeps track of only those tuples in R, which have been processed against l and have the k highest similarities with l. In the HHNL algorithm, and also in all other algorithms described in that study, a heap structure is used to find the smallest of the k largest similarities.

The HVNL (Horizontal-Vertical Nested Loops) algorithm is an adaptation

of the ranked query evaluation techniques in the IR domain to the join operation [59]. In an IR system, the aim is to find the k documents in the system which are most similar to the user query. For that purpose, most of the IR systems employ inverted indexes. In these systems, for each term t in the user query, the term is searched from the inverted index and the ids of documents containing term t are found. Then, the similarity calculations are performed only for those documents that have at least one common term with the user query. Algorithm HVNL is a straightforward extension of this method such that for each tuple l in L, the algorithm calculates the similarity of l to all tuples in R having at least one common term with l, and selects the k most similar tuples from R. The advantage of HVNL algorithm is that, it does not perform similarity calculations for all tuples in R as in the case of the HHNL algorithm. In the HVNL algorithm, the inverted index is in the memory, the inverted list entries, and the relations L and R are read from disk.

The algorithm VVM (Vertical-Vertical Merge) employs sorted inverted indices with respect to the index terms on both of the input relations L and R [59]. The VVM algorithm scans both inverted files on the input relations at the same time. During the scan of the inverted indices, if both index entries correspond to the same index term, then similarities are accumulated between all tuples in the inverted lists of the indices. The VVM algorithm assumes that, both inverted files as well as relations L and R are read from disk. In order to store intermediate similarities between every pair of tuples in the two relations, the algorithm needs accumulators for $|L| * |R|$ tuple pairs that are stored in main memory. The strength of the algorithm is that it scans the inverted files only once to compute similarities between every pair of tuples. However, the memory requirement for the accumulator[4] is so large that it cannot be run for relations having large number of tuples. Assume that both relations L and R consist of 100,000 tuples, and each similarity value requires 4 bytes (size of float), so the memory allocated for the accumulator should be at least 100,000*100,000*4 bytes = 40Gb.

[4]An accumulator for a tuple l is a set of real numbers each of which is used to store the accumulated similarity value between tuple l and tuples r from R. For each new tuple r considered for similarity comparison, a new element is inserted to the accumulator.

Cohen developed a database management system called WHIRL (Word-based Heterogeneous Information Representation Language), which supports text similarity based joins [25]. WHIRL's text similarity based join algorithm makes reasoning about the similarity of the join attributes of the tuples l in relation L and r in relation R. The output of the join operation is a set of ordered tuples so that the "best" tuple pairs are presented first. WHIRL considers tuple pairs to be "better" when the similarity conditions required by the join operation are more likely to hold. WHIRL's algorithm finds the highest scoring tuples without generating all low scoring tuples. It makes heavy use of inverted indexes and tf-idf weighting scheme. It is assumed that, every tuple $< \vec{v}_1, \vec{v}_2, ..., \vec{v}_m >$ has exactly m components, and each of these components is a text fragment, represented as a document vector over the vocabulary. It is also assumed that a "score" is associated with every tuple. This score is always between 0 and 1, and it measures the degree of belief in a fact.

In WHIRL, finding best tuples is viewed as an optimization problem; in particular, the join algorithm uses A* search method to find the highest scoring k substitutions for joining tuples. The goal is to find a "small" number of "good" substitutions. The search method uses some short-cut evaluation techniques from IR ranked retrieval. For example, tuples containing high weighted query terms are searched. If a tuple does not contain a high weighted search term, then it is not possible that the tuple has high similarity to the search tuple. The current implementation of WHIRL keeps all indices and document vectors in main memory [25].

WHIRL's similarity based text join algorithm finds k most similar tuples from relation R to an input tuple l as follows: Initially, r tuples containing the highest weighted term in tuple l are found. The score of each r tuple is the similarity of that tuple to tuple l, and this value is simply calculated by multiplying the document vectors of tuples l and r. After finding scores for all tuples that contain the highest weighted term, the algorithm computes the maximum score for r tuples (i.e., $f(r'_j)$ value) which do not contain the highest weighted term. If the score of a tuple i containing the highest weighted term is greater than the $f(r'_j)$ value, the algorithm outputs that tuple, otherwise it

continues with the next highest weighted term. As the algorithm uses A* search, it does not compare every tuples from R with tuple l; it only compares tuples which have high probability to have high similarity to tuple l.

In addition to Meng et al. [59] and Cohen's [25] join algorithms, Gravano et al. also studied the text join operation in a relational database management system (RDBMS) [36]. They described a technique for performing text join operation within an unmodified RDBMS using plain SQL statements. In their work, tf-idf weighting scheme and the cosine similarity metric are employed to identify potential string matches. Two strings are joined if their similarity exceeds a specified threshold. To perform the join operation, a sampling based strategy using the cosine similarity metric implemented with pure SQL queries is developed, however, no specialized operator is designed for the join operation.

5.3 Text Similarity Based SVA Directional Join Algorithms

In this thesis, we present a new algorithm for the implementation of the text similarity based SVA directional join operator. Our algorithm is an improved and extended version of the algorithms presented in [59]. We also employ some early termination heuristics from the IR domain [81] for further possible improvement of the performance of our algorithm. In this section, we introduce our new algorithm and then, we propose extensions to our algorithm involving heuristics from the IR domain.

In [59], the tuples to be joined consist of arbitrarily long text documents. In our work, however, we assume that the join attribute of tuples contains short strings (e.g., person name, company name, book title, etc., which consist of at most 50 words) as in the text similarity join algorithm of WHIRL. When the join attribute consists of only short texts, it is easier to create an inverted index on the input relation, and the join algorithm does not require an a-priori generated inverted index as an input. We designed an algorithm, named Inverted Index

based Nested Loop (IINL), which takes only two relations L and R as input, and constructs an in-memory inverted index for one block of tuples from relation R that are currently in the memory during the join operation. Our algorithm is presented in Figure 5.1.

The IINL algorithm is a block nested loops join algorithm, in which while there are unprocessed tuples in relation L, one block of tuples from L are read, and for each block of tuples read from L, the algorithm starts reading tuples from the beginning of the relation R. For each block of tuples read from R, an in memory inverted index is created. As the inverted index is constructed only for one block of tuples, it does not require huge amount of memory space and it can fit into the main memory. We discuss the amount of memory required for the inverted index in Section 5.4. After being constructed, the inverted index is used for similarity computations. As the inverted is constructed for one block of tuples from relation R, we assume that all the statistical data about the relation R (e.g., number of tuples in R, frequency of each term in relation R, etc.) that are necessary for the similarity computations are known in advance by just making single pass over the relation R as pre-processing step which takes only a few seconds.

With the IINL algorithm, we intend to make use of the benefits of both HHNL and HVNL algorithms. The HHNL algorithm, as we discussed in the previous section, is a blind nested loops join algorithm and it reads blocks of tuples from its input relations and compares every tuple pairs. The advantage of the HHNL algorithm is it makes only $\lceil(|L|/Block\_size(L))\rceil + \lceil(|L|/Block\_size(L))\rceil * \lceil(|R|/Block\_size(R))\rceil$ disk accesses, but $|L| * |R|$ tuple comparisons. The HVNL algorithm, on the other hand, employs an inverted index on relation R to make very few number of tuple comparisons, however for each inverted list entry that is not in the memory it has to access to the disk. As we employ in-memory inverted index in the IINL algorithm, it does not need to access to the disk for retrieving inverted list entries and it makes only $\lceil(|L|/Block\_size(L))\rceil + \lceil(|L|/Block\_size(L))\rceil * \lceil(|R|/Block\_size(R))\rceil$ disk accesses and does very few number of tuple comparisons as we discuss in the next section. The only drawback of the IINL algorithm is that it needs to create an in-memory inverted index for

Algorithm: IINL

Input: Relations L and R, an importance score propagation function f_{out}, an integer k.
Output: Joined tuples with respect to importance scores.
Var: An accumulator A.

begin (IINL)
while there are unprocessed tuples in relation L **do**
◇ Read X tuples from L. (If the number of the unprocessed tuples is less than X, read all the unprocessed tuples into the main memory.)
◇ Go to the beginning of relation R.
while there are unprocessed tuples in relation R **do**
∘ Read Y tuples from R. (If the number of the unprocessed tuples is less than Y, read all the unprocessed tuples into the main memory.)
∘ Create an in-memory inverted index on tuples that are being read.
for each unprocessed tuple l in L in the memory **do**
for each term t in l **do**
Search t from the inverted index.
if t is found **then**
Accumulate similarities between l and all the tuples in the inverted list entry for term t (I_t).
end if
end for
for each accumulated similarity ($A_r \in A$) for tuple l **do**
Compute the derived importance score as $f_{out}(l.score, r.score) * A_r$.
if the derived importance score is greater than the smallest of the k largest scores computed so far for l **then**
∘ Replace the smallest of the k largest scores by the new score.
∘ Update the list (S_l) of tuples in R to keep track of those tuples with the k largest scores with l.
end if
end for
end for
end while
◇ Join each l tuple in the memory with R tuples in the list S_l.
end while
end (IINL)

Figure 5.1: The IINL algorithm.

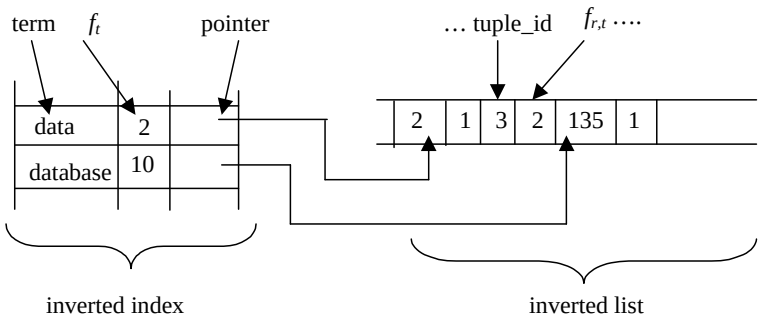

Figure 5.2: Inverted index structure.

each Y tuples read from R. However, this task is not costly, since the join attribute consists of short texts, and making only a single pass over the R tuples in the memory is enough to create the index, as we discuss in Section 5.4. Also, once the inverted index is constructed for each block of R tuples, it can be written on the disk and it can be re-used in the subsequent similarity computations, again this does not increase the number of disk accesses considerably (see Section 5.4). The HVNL algorithm, on the other hand, requires a pre-constructed inverted index on relation R as input to the algorithm.

The inverted index employed in the IINL algorithm has the structure shown in Figure 5.2. The inverted index is an array of records and it contains the index term (t), the number of tuples in R containing the index term (f_t), and a pointer to its corresponding inverted list entry. The inverted list is also an array of records and it stores tuple id containing the index term t, and the frequency of the term in that tuple ($f_{r,t}$). As an example, according to the Figure 5.2, the index term "data" occurs in two tuples having ids 2 and 3, and tuple 2 contains the term only once, while in tuple 3 the frequency of the term "data" is 2.

The use of the inverted index and the inverted list in similarity based text join for both the IINL algorithm and the HVNL/VVM algorithms can be explained as follows. Let t be the term in tuple l to be considered, and let $w_{l,t}$ be its weight. Let the inverted list entry for the term t be $\{(r_1, f_{r_1,t}), ..., (r_n, f_{r_n,t})\}$ where r_i is the tuple id and $f_{r_i,t}$ is the frequency of term t in tuple r_i. The weight $w_{r_i,t}$ for term t for tuple r_i can be calculated using $f_{r_i,t}$, and term weights for L tuples are computed as we explained in Section 5.1. After term t is processed, the similarity

between l and r_i accumulated so far will be $A_{r_i} + w_{l,t} * w_{r_i,t}$, where A_{r_i} is the accumulated similarity between l and r_i before t is considered, and $w_{l,t} * w_{r_i,t}$ is the contribution due to the sharing of the term t. After all terms in tuple l are processed, the similarities between l and all the tuples in R will have been computed and stored in A_{r_i}, and the derived importance scores are computed by $f_{out}(l.score, r_i.score) * A_{r_i}$ and the k tuples in R having the highest derived importance scores can be identified.

As we employ inverted index during the similarity computations of the IINL algorithm, some early termination heuristics [81] from the IR domain can be used to decrease the similarity computations made during the join operation. Applying early termination heuristics reduces the search space for finding similar tuples, and decreases the number of tuple comparisons to be done, that's why they are called "early termination" heuristics. These heuristics improve the performance of the join operation by considering only the R tuples that have high similarity to a given tuple from L. The main idea behind the early termination heuristics is that if a tuple r from relation R has high weighted terms of tuple l of relation L, then it is more likely that tuple r has higher similarity to tuple l than the other tuples from R that do not contain high weighted terms. We discuss these heuristics in more detail in the sections below, and present different variations of the IINL algorithm employing these heuristics.

The IINL algorithm is a nested-loops based join algorithm. Other join techniques, sort-merge and hash join, are not suitable for text similarity based SVA directional join operator. Sort-merge is not applicable to our join operator since our input relations are not sorted. Even if the input relation R is sorted, the performance of the sort-merge algorithm would not be better than the nested-loops based algorithms, because our join condition is a similarity based condition and it requires multiple passes over the input relation R. Since the join condition is not an equality condition, hash join algorithm is also not applicable to our join operator.

5.3.1 Harman Heuristic

Harman et al. [39] proposed a heuristic to decrease the number of similarity computations performed during the search of similar documents to a user query. According to this heuristic, the terms in the given user query are sorted with respect to their weights in descending order, and only the terms whose weights are greater than the 1/3 of the highest weighted term in the query are considered. Then, the inverted list is accessed only for these terms and in decreasing weight order.

The IINL algorithm extended with Harman heuristic is called IINL-Harman. In this variation of the IINL algorithm, for each tuple l in relation L, weights of the terms in l are examined, and the in-memory inverted index is accessed only for these terms having a weight greater than the 1/3 of the highest weighted term in the tuple. The IINL-Harman algorithm considers R tuples which have high weighted terms in tuple l, and does not perform similarity computations for other R tuples that do not contain high weighted terms.

5.3.2 Quit and Continue Heuristics

Moffat et al. [63] also suggested to sort the terms in the user query with respect to their weights in descending order, and access the inverted index with respect to this order. They place an a priori bound on the number of nonzero accumulators permitted. In other words, they place a bound on the number of candidate documents that can be considered for the similarity calculation. New accumulators are added until this bound is reached. The idea behind this heuristic is that, terms of high weight are permitted to select accumulators, but terms of low weight are not allowed to contribute to the similarity computation. When the accumulator bound is reached, then there are two possibilities: In the *quit* approach, the cosine contribution of all unprocessed terms are ignored, and the accumulator contains only partial similarity values for documents. In the *continue* strategy, documents that do not have an accumulator are ignored, but documents for which accumulators have already been created continue to have their cosine

contributions accumulated. When the processing ends, the computation of full cosine values for a subset of the documents becomes completed.

As the quit heuristic allows only the partial similarity computation, it is not suitable for the text similarity based SVA directional join operator. In order to find top k similar tuples having highest derived importance scores for a given tuple l in relation L, we need to have full cosine values. Thus, we use the continue heuristic with our IINL algorithm (IINL-Continue). In this variation of the IINL algorithm, for each tuple l of L, only r tuples from R which have high weighted terms in l are considered for similarity computations until the accumulator bound (i.e. an upper bound on the number of tuples that can be considered for similarity computations) is reached. Then, r tuples for which accumulators have already been created continue to have their cosine contributions accumulated. When the processing ends, the full cosine similarity between tuple l and r becomes computed, and the importance scores are derived by the $f_{out}(l.score, r.score) * A_r$ value, and the k similar and having highest derived importance score r tuples for l tuple are selected.

In the IINL-Continue algorithm, to select r tuples having high importance scores during similarity computations (until the accumulator bound is reached), the inverted list entries of the in-memory inverted index is sorted in descending order of the importance score values of r tuples. As the inverted list entries are kept in the memory, this sort operation is not costly.

5.3.3 Maximal Similarity Filter

"Maximal similarity filter" is another technique that may be used to reduce the number of tuple comparisons made during the text similarity based SVA directional join operation [69]. Let $u_r =< u_1, u_2, ...u_n >$ be the normalized term vector corresponding to the join attribute of tuple r of R, where u_i represents the weight of the term i in the join attribute. Assume that the filter vector $f_L =< w_1, w_2, ...w_x >$ is created such that each value w_i is the maximum weight of the corresponding term i among all vectors of L. Then, if $\cos(u_r, f_L) < V_t$

then r can not be similar to any tuple l in L with similarity above V_t. The value $\cos(u_r, f_L)$ is called the *maximal similarity* of a record r in R to any other record l in L. The maximum value of a term for a given relation is determined while creating the vectors for the tuples, and the filter vector for each relation may be formed as a one-time cost.

In the IINL algorithm with maximal similarity filter (IINL-Max-Filter), the in-memory inverted list entries are sorted with respect to descending order of $\cos(u_r, f_L) * r.score$ (i.e., importance score of tuple r revised with its maximal similarity value) for r tuples. For each term t in tuple l of L, the inverted index is entered and the similarity comparisons are stopped at the point when the importance score revised with maximal similarity value ($\cos(u_r, f_L) * r.score$) for the tuple r is less than the smallest of the k largest derived importance scores ($f_{out}(l.score, r.score) * \cos(u_l, u_r)$) computed so far for tuple l, since it is not possible for r to be in the top k similar and having high derived importance score tuples list.

The maximum weight of a term for a given relation L is determined while creating the vectors for the tuples, and the filter vector for each relation may be formed as a one-time cost. To improve the performance of the IINL-Max-Filter algorithm, inverted list entries are sorted with respect to importance scores revised with maximal similarity values of tuples in descending order, and this operation is not costly as the inverted index is stored entirely in the memory.

5.3.4 Other Improvements

In addition to the above early termination heuristics, the following well-known IR heuristic can also be employed: store term weights ($w_{r,t}$) in the inverted index instead of term frequency ($f_{r,t}$) values since all values necessary for calculating the weight are known in advance [82]. Also, during the implementation of all of the above algorithms, we observed that, we may preprocess the relation L in such a way that it contains the inverted index address for each term instead of the term itself in each tuple. So, we can directly access the inverted index without making

any search over the index. Another observation made is that, in the similarity join operation, we determine similarity of short strings like paper titles, company names, author names, etc., and we do not need to consider term frequency for each term in the tuple due to the fact that term frequency for almost all of the terms in a tuple is 1. When we plug 1 for $f_{r,t}$ value in the term weight ($w_{r,t}$) formula, $w_{r,t}$ value becomes 1. Then the cosine similarity equation reduces to

$$\cos\_sim(l,r) = \frac{1}{W_r} \sum_{t \in (l \cap r)} w_{l,t} \tag{5.6}$$

and evaluating this similarity equation is cheaper than evaluating the original formula.

All these observations may serve to improve the running time performance of the algorithms described in the above sections. However, for the experimentation, we implemented the HHNL, HVNL, WHIRL's algorithm, and all versions of the IINL algorithm, and compared these algorithms with respect to the number of tuple comparisons, and the number of disk accesses made. As the improvements mentioned in this subsection does not serve to improve the number of tuple comparisons and disk access made, we do not include them in our experiments. The experimental results are provided in the next section.

5.4 Experimental Results

We compare the performance of our new algorithm with the ones developed previously [25, 59] in terms of the number of tuple comparisons made and the number of disk accesses required. For the experimentation, all of the algorithms (HHNL, HVNL, WHIRL's algorithm, and all versions of the IINL algorithm) except the VVM algorithm were implemented in C programming language under MS Windows98 operating system. During the implementation, we extended the HHNL, HVNL, and WHIRL's algorithm with score management functionality, and implemented a disk based version of WHIRL's algorithm in which the inverted index is in-memory, but the inverted list entries are stored on disk. We did not include VVM since it requires huge amount of memory to keep intermediate similarities

between tuple pairs.

We used a real dataset that consists of the bibliographic information of journal and conference papers obtained from the DBLP Bibliography database [51]. In the experimentation, the relations L and R do not contain any common tuple, and the relation L consists of bibliographic information of approximately 90,000 journal papers, and the relation R contains bibliographic information of 132,000 conference papers. We take the paper title attribute as the join attribute, and for each journal paper l in relation L, we try to find k conference papers from relation R having similar titles to the title of l and have highest derived importance scores. We created the vectors for the join attribute of each tuple in the relations L and R, the maximal similarity filter vector for relation L, and the inverted index on relation R in advance. We assumed that input relations L and R, and the inverted lists are stored on disk, and we have enough main memory to store the inverted index and the accumulators used for similarity calculations. The experiments were performed on a PC having Pentium III 450 MHz CPU and 320 MB of main memory.

5.4.1 Tuple Comparisons

Some of the experimental results for the implemented algorithms are provided in Figure 5.3 through Figure 5.6. In Figure 5.3 and Figure 5.4, the number of tuple comparisons performed by all of the implemented algorithms during the text similarity based SVA directional join operations for different k values are presented.

As shown in Figure 5.3, the HHNL algorithm needs to make around 12 billion comparisons for each different k values to join L and R, while all of the other algorithms do less than 900 million tuple comparisons for the same join operation (Figure 5.4). The HVNL, WHIRL, and all variations of the IINL algorithm perform much better than the HHNL algorithm, because of the fact that these algorithms employ inverted index on the input relation R, and they only perform similarity calculations for tuple pairs having at least one common term; in other

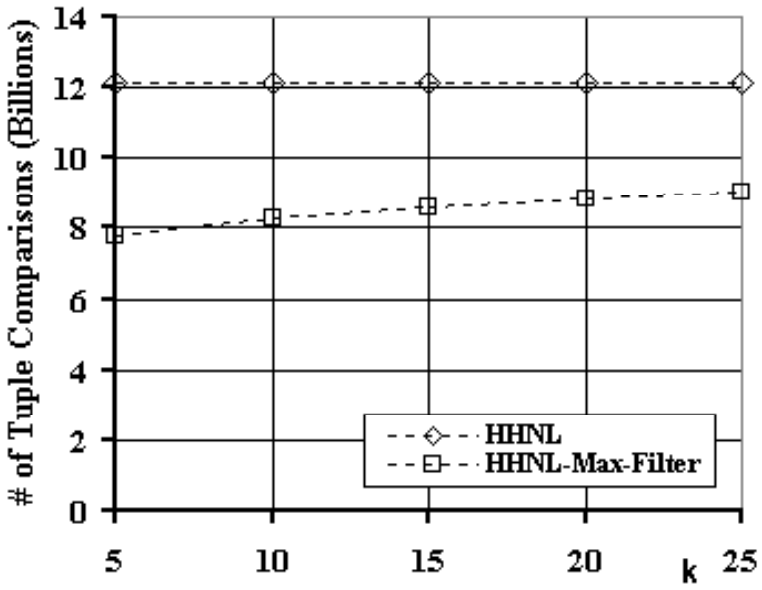

Figure 5.3: Number of tuple comparisons required by the HHNL algorithm for different k values.

words, these algorithms compare similarity of tuples which are guaranteed to have a similarity value greater than 0. The HHNL algorithm, on the other hand, makes similarity computation for all tuple pairs regardless of whether the tuples contain any common term or not.

The HHNL-Max-Filter algorithm in Figure 5.3, is a variation of the HHNL algorithm, which employs the maximal similarity filter heuristic to reduce the number of tuple comparisons to be performed. As shown in Figure 5.3, the maximal similarity filter heuristic, by itself, reduces the number of tuple comparisons by approximately 25%.

In Figure 5.4, the number of tuple comparisons required by the HVNL, WHIRL, and all variations of the IINL algorithm for different k values are displayed. The IINL and HVNL algorithms make exactly the same number of tuple comparisons, and the performance results of the IINL-Harman, and WHIRL algorithms are almost the same as the IINL and HVNL algorithms. The IINL algorithm employing the maximal similarity filter heuristic, makes about 25% less number of tuple comparisons with respect to the IINL algorithm, and the

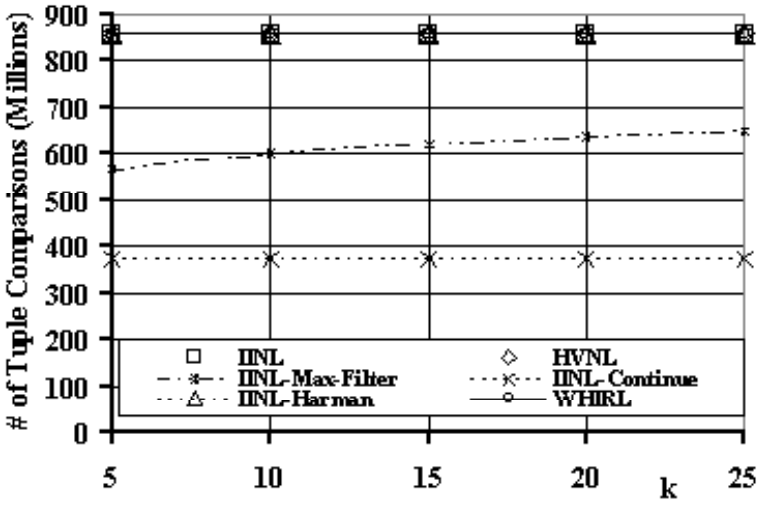

Figure 5.4: Number of tuple comparisons required by the HVNL, WHIRL and IINL algorithms for different k values.

continue heuristic[5] provides more improvement on the performance of the IINL algorithm by decreasing the number of tuple comparisons by 50%. The Harman heuristic does not improve the performance of the IINL algorithm, because term weights for our input data are quite close to each other. The Harman, continue, and maximal similarity filter heuristics make the same percentage of reduction in the number of tuple comparisons performed by the other algorithms. However, the Harman and continue heuristics can only be used in algorithms that employ inverted index (i.e., HVNL and WHIRL), the maximal similarity filter heuristic, on the other hand, can be applied to all algorithms described in this chapter.

5.4.2 Disk Accesses

Figure 5.5 displays the results obtained in terms of the number of disk accesses required by the HHNL, HVNL, WHIRL, and the IINL algorithms when the relations L and R, and the inverted list entries on the join attribute of relation R for the HVNL and WHIRL algorithms are stored on disk. We assumed that the HHNL and IINL algorithms read 10,000 tuples from the relations L and R at

[5]We chose the accumulator bound for the continue heuristic as 5000 tuples for this experiment. In the subsequent sections, we present the experimental results that show the effect of the accumulator bound on the join algorithm performance.

each iteration of the inner and outer loops; the HVNL and WHIRL algorithms read 10,000 tuples at a time from the relation L, and store upto 5,000 inverted list entries in the memory. For each term t considered during the similarity comparisons, the HVNL and WHIRL algorithms read inverted list entries of term t (i.e., I_t) only. If I_t is not in the memory and there is no available space for I_t in the memory, the inverted list entry which is currently held in the memory for a term having the least term frequency, is replaced with I_t. Under these assumptions, we observed that, the number of disk accesses performed by the HHNL and all versions of the IINL algorithms, which is approximately 150 disk accesses, are quite less than those obtained with the HVNL and WHIRL algorithms. As the IINL algorithm creates an in-memory inverted index for the similarity computations, it does not need to make disk accesses for retrieving the inverted list entries, and thus, the early termination heuristics employed in the IINL algorithm can not reduce the number of disk accesses performed. So, we did not give the performance results for the IINL-Harman, IINL-Continue, and IINL-Max-Filter algorithms in Figure 5.5. The number of disk accesses required by the HVNL and WHIRL algorithms are much higher than that with the HHNL algorithm, although the number of tuple comparisons performed is much lower. This result is due to the fact that the HVNL and WHIRL algorithms need to make at least one disk access to read inverted list entries that are not in the main memory, and they access the inverted list for almost all terms in each tuple of relation L.

In IINL algorithms, the in-memory inverted index reduces the number of disk accesses significantly, however it incurs the extra cost of index construction during the join operation. For each block of tuples (10000 tuples, each of them includes at most 50 terms) that are read from the relation R, 27 seconds is required to create the in-memory inverted index which is only 10% of the time required to perform the join operation. If we write the in-memory index on disk to be re-used whenever necessary during the join operation, it takes just 1 second to write and read the inverted index. As the IINL algorithm is a nested loops based algorithm, we read the same blocks of the inner relation multiple times, and we do not need to create inverted index for the blocks at each time. Once the inverted indexes over the blocks are written on the disk, they can also be used for the subsequent

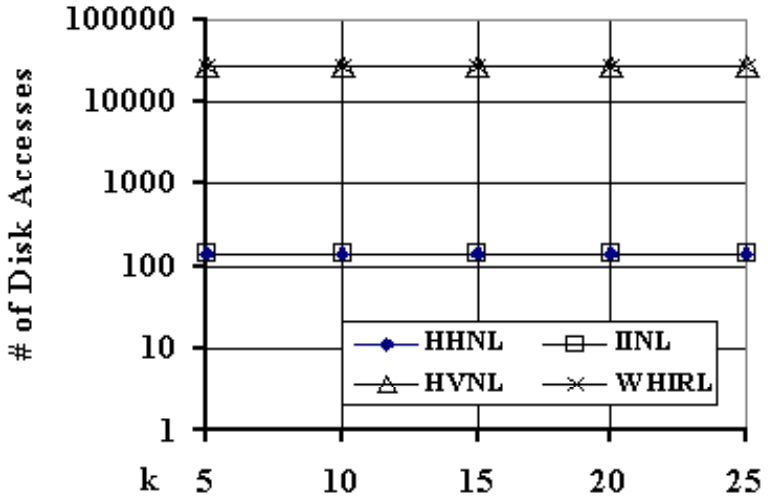

Figure 5.5: Number of disk accesses performed by all the similarity join algorithms for different k values.

join operations performed over the same relation. Writing the in-memory index on disk and reading it during the join operation increases disk accesses only by $\lceil(|L|/Block\_size(L))\rceil * \lceil(|R|/Block\_size(R))\rceil$ which equals 140 disk accesses for our data set, and this value is considerably less than the number of disk accesses performed by the HVNL algorithm (see Figure 5.5). In the HVNL algorithm, the creation of inverted index over the whole R relation requires 8764 seconds, which is much higher than the index creation cost of the IINL algorithm.

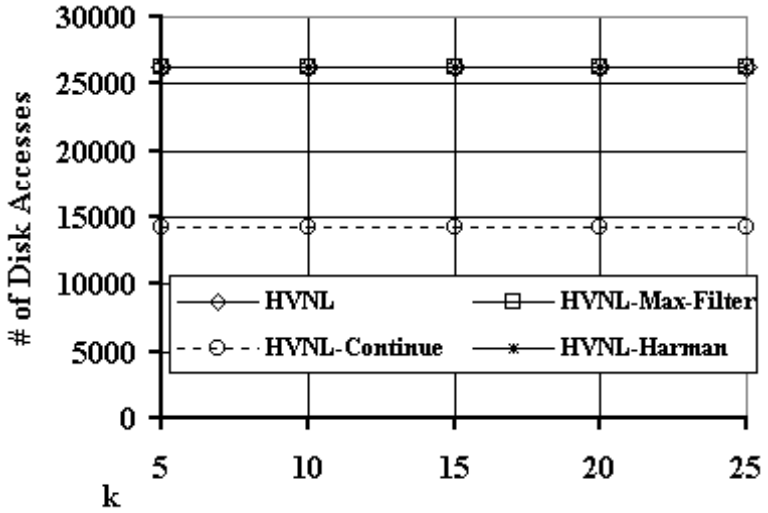

Figure 5.6: Number of disk accesses performed by the early termination heuristics for different k values.

We performed another experiment to measure the reduction made by the early termination heuristics on the number of disk accesses required when the inverted list entries are stored on disk. For this experiment, we implemented the HVNL algorithm with early termination heuristics namely the Harman, continue, and maximal similarity filter heuristics. We assumed that we have an inverted index on relation R, we have enough memory to store the inverted index, and the inverted list entries are stored on disk. Also we assumed that, for the continue heuristic the accumulator bound is 5000 tuples. The results of this experiment are displayed in Figure 5.6. According to this figure, the continue heuristic reduces the number of disk accesses of the HVNL algorithm by 50%. The Harman and maximal similarity filter heuristics, on the other hand, do not lead to any reduction on the number of disk accesses required. This result is due to the fact that, the term weights in our dataset are close to each other and the Harman heuristic considers almost all terms in a tuple l during the similarity computations. The maximal similarity filter heuristic on the other hand, needs to access all the inverted list entries for all terms in a tuple l to find the r tuples having high importance scores revised with maximal similarity values. Therefore, this heuristic only reduces the number of tuple comparisons performed when the inverted list entries are sorted with respect to the importance score revised with maximal similarity value of tuples.

In the continue heuristic, the accumulator bound is an important factor on the performance of the join algorithm. To show the effect of the accumulator bound on the join operation, we run the HVNL-Continue algorithm with different accumulator bounds. We observed that, the number of tuple comparisons and the number of disk accesses decreases as the accumulator bound is decreased as shown in Table 5.1. This is due to the fact that the accumulator bound is an upper bound on the number of tuples that can be considered for similarity comparisons. The number of disk accesses and the tuple comparisons made remain the same for different k values.

Table 5.1: The effect of accumulator bound for the continue heuristic on the number of tuple comparisons and disk accesses made, and the accuracy of the join operation.

Accumulator Bound	**# of Tuple Comparisons**	**# of Disk Accesses**	**Accuracy**
5,000	372,448,481	14,176	65%
10,000	604,454,778	20,001	84%
15,000	732,112,934	22,678	91%

5.4.3 Accuracy of the Early Termination Heuristics

We calculated the accuracy of the output produced by the algorithms that employ early termination heuristics as follows: $Accuracy = |B \bigcap H|/|B|$. In this equation B denotes the output set generated by any one of the HHNL, HVNL, or IINL algorithms as all of these algorithms produce the same output, H is the output set generated by one of the IINL or HVNL algorithms that employ any one of the early termination heuristics, and $|.|$ denotes the set cardinality. We chose the output set of any one of the HHNL, HVNL, and IINL algorithms as the base for the accuracy calculation, because these algorithms do not miss any pairings that must be in the result set. The Harman, and the continue heuristics compare tuple pairs which have high probability to have high similarity, thus it is possible that they may not find all of actual top k similar tuples. So, we defined the accuracy of a heuristic as the percentage of the "true match"es that are generated by that heuristic, and calculated the accuracies by using the above formula. We observed that the Harman heuristic generates exactly the same output as the HHNL, HVNL, and IINL algorihms; the continue heuristic, on the other hand, could achieve 65% accuracy when the accumulator bound is set to 5000, and the accuracy can be improved to 91% when the accumulator bound is increased to 15000, as shown in Table 5.1. As the accumulator bound is increased, the accuracy of the continue heuristic also increases, since the continue heuristic allows more tuples to be considered during the similarity comparisons. For the maximal similarity filter heuristic, we observed that the accuracy of this heuristic is 100%, as it calculates the similarity for r tuples having importance score revised with maximal similarity value greater than or equal to the smallest of the k largest

derived importance scores computed so far for tuple l. Therefore, the heuristic considers all r tuples that can be in the result set by eliminating the ones that are not possible to be in the result.

5.4.4 Memory and CPU Requirements

We also compared the HHNL, HVNL, WHIRL, and IINL algorithms in terms of the amount of memory they required. As the HHNL algorithm is a block nested loops based join algorithm, it requires lb bytes to keep one block of tuples from relation L, rb bytes for one block of tuples from relation R, and $|LB| * h$ bytes where $|LB|$ is the number of L tuples that are currently in the memory and h is the size of a heap having k elements to keep the k largest derived importance scores computed so for a tuple in L that is currently in the memory. Thus, the HHNL algorithm requires at least $lb+rb+|LB|*h$ bytes of memory. In the HVNL algorithm, we need $lb + nx + nl + acc$ bytes where lb bytes is required to read one block of tuples from relation L, nx bytes to store the inverted index, nl bytes to keep inverted list entries for m number of index terms, and acc bytes for the accumulator. The memory requirement of the WHIRL algorithm is very similar to that of the HVNL algorithm. For the WHIRL algorithm, we need lb bytes for one block of L tuples, nx bytes for the inverted list, nl bytes for the m number of inverted list entries, os bytes for the set of open states, and ex bytes for the list of exclusion terms. The IINL algorithm needs lb and rb bytes for one block of tuples from the relations L and R, respectively, nxl bytes for the in-memory inverted index and the inverted list entries for the R tuples that are currently in the memory, acc bytes to accumulate similarities between a tuple l and r tuples in the memory, and $|LB|*h$ bytes to keep the k largest derived importance scores computed so far for each L tuple that is currently in the memory. In Table 5.2, we give all the variables in the memory requirement formulas for the HHNL, HVNL, WHIRL, and IINL algorithms, and their values for the L and R relations generated from the DBLP Bibliography data.

As presented in Table 5.2, the minimal memory requirements of the algorithms are as follows: the HHNL algorithm requires 90MB, the HVNL algorithm

Table 5.2: Statistical data for the L and R relations obtained from the DBLP Bibliography data.

Variable	**Explanation**	**Value**
$\|L\|$	# of tuples in relation L	91,230
$\|R\|$	# of tuples in relation R	132,748
$\|LB\|$	# of tuples from L that are currently in the memory	10,000
$\|RB\|$	# of tuples from R that are currently in the memory	10,000
t	size of a tuple (in bytes)	4372 bytes
lb	size of a block of tuples from L (in bytes) $= t * \|LB\|$	43,720,000 bytes
rb	size of a block of tuples from R (in bytes) $= t * \|RB\|$	43,720,000 bytes
h	size of a heap to keep k largest similarity values and corresponding tuple ids for a tuple from L	300 bytes
V	# of distinct terms in relation R	30,625
x	size of an inverted index entry	83 bytes
nx	size of the inverted index $(x * V)$	2,541,875 bytes
i	size of an inverted list entry	28 bytes
I	# of inverted list entries	809,497
inl	size of the inverted list $(i * I)$	22,665,916 bytes
m	# of index terms whose inverted list entries are kept in the memory	5,000
nl	average size of the inverted list that are kept in the memory for the HVNL algorithm $(m * i * I/V)$	3,700,557 bytes
acc	maximum size of the accumulator $(\|R\| * 8$ bytes)	1,061,984 bytes
os	average size for storing the open states in the WHIRL algorithm $(\|r\| * (I/V) * 12$ bytes)	15,859 bytes
ex	maximum size for the list of exclusion terms $(\|r\| * term\_size)$	3,750 bytes
$\|r\|$	maximum # of terms in a tuple	50
$term\_size$	maximum size of a term (in bytes)	75 bytes
nl'	average size of the in-memory inverted list for the IINL algorithm $(i * I * \|RB\|/\|R\|)$	1,707,439 bytes
nxl	size of the in-memory inverted index and inverted list for the IINL algorithm $(nl' + nx)$	4,249,314 bytes

needs 51 MB, the WHIRL algorithm has need of 52 MB, and the IINL algorithm necessitates 96 MB of memory. The IINL algorithm requires more memory space with respect to the other algorithms; however, the extra memory space required may not be considered to be large for today's computer technology.

In order to estimate the CPU requirement of the HHNL, HVNL, WHIRL, and IINL algorithms, we considered the number of similarity computations (i.e., tuple comparisons) performed during the join operation, as the similarity computation is the most expensive operation. In Section 5.4.1, we provide the number of tuple comparisons made by each algorithm during the join of the relations L and R. Here, we try to find an upper bound on the number of tuple comparisons done by the HHNL, HVNL, WHIRL, and IINL algorithms. As the HHNL algorithm is a block nested loops join algorithm, it compares every tuple pair from the relations L and R, and makes exactly $|L| * |R|$ tuple comparisons. The HVNL, WHIRL, and IINL algorithms employ an inverted index to reduce the similarity computations performed during the join operation, and these algorithms perform similarly to each other as it is shown in Figure 5.4. In any text similarity based SVA directional join algorithm employing an inverted index, for each term in a tuple l, the term is searched from the inverted index and the tuple l is compared with every tuple in the inverted list entry for that term. The upper bound on the number of tuple comparisons can be calculated as follows:

$$\sum_{i=1}^{n}(f_{t_i,L} * f_{t_i,R}) \tag{5.7}$$

where t_i is a term in relation R, $f_{t_i,L}$ is the frequency of term t_i in relation L, $f_{t_i,R}$ is the frequency of term t_i in relation R, and the frequency of each term in a relation can be estimated using the Zipf's law [9].

5.5 Discussion

Employing inverted index during the similarity computations of the similarity based directional text join operator reduces considerably the number of tuple comparisons made (e.g., from 12 billion comparisons to 900 million comparisons

as shown in Figures 5.3 and 5.4). However, storing the inverted list entries on disk increases the disk accesses required during the join operation (e.g., from 150 disk accesses to 26,000 disk accesses). In order to make use of the benefits of employing an inverted index during the similarity computations while experiencing very small number of disk accesses, we developed a new algorithm named Inverted Index based Nested Loop (IINL). As we used inverted index during the similarity comparisons, we were able to employ some early termination heuristics (i.e., the Harman, continue, and maximal similarity filter heuristics) into our text similarity based SVA directional join algorithm, and observed a reduction in the number of tuple comparisons experienced. We showed in Sections 5.4.1 and 5.4.2 that, the continue and the maximal similarity filter heuristics reduce the number of tuple comparisons and the disk accesses significantly; however, the Harman heuristic, does not provide any performance improvements. The performance of the continue heuristic is highly dependent on the value of the accumulator bound. As the accumulator bound decreases, the number of tuple comparisons and the number of disk accesses decrease, however the accuracy of the output also decreases. The IINL algorithm with continue heuristic (IINL-Continue) may be used when fast response to the join operator is needed. Otherwise, the IINL-Max-Filter performs best in terms of the number of tuple comparisons and the disk accesses performed compared to all the other algorithms.

Chapter 6

Performance Evaluation

As we propose a Web querying system in this thesis, we conducted an experiment to measure the search effectiveness of our system against the state-of-the-art keyword-based search techniques (e.g., boolean, ranked retrieval) that are widely used in search engines. For the experimentation, we used two metadata databases, namely Stephen King metadata and DBLP Bibliography metadata. Stephen King metadata is a small sized metadata database that provides advice over 280 Web documents about the famous horror novelist Stephen King and his books. DBLP Bibliography metadata database, on the other hand, contains metadata about more than 225,000 computer science research papers that are located at the DBLP Bibliography server [51]. A more detailed information about the metadata databases are given in Section 6.2.

We designed and run two groups of test queries over the two expert advice databases. The first group of queries requires our specialized operators (e.g., text similarity based selection, text similarity based directional join, topic closure) to be answered correctly. We run these queries over the two metadata databases and compute the precision of the output and running time of the queries. Our aim in this part of the experiment is to show that, our SQL extensions allow us to form useful queries that can not be formulated by boolean and ranked queries, and our SQL extensions generate high precision query outputs in a reasonably short amount of time.

The second group of queries do not require any specialized operator, they can be formulated in plain SQL, and also they can be represented as boolean and ranked queries. We again run these queries over the same metadata databases. Also, we use inverted indexes created over the Web documents that are covered by the two metadata databases. We run the boolean and ranked queries over these inverted indexes, and compare the precision of the query outputs generated by querying the metadata databases against boolean and ranked querying of the inverted indices. In this part of the experiment, our aim is to compare the precision of the metadata-based search with traditional keyword based search techniques.

In Section 6.1 we explain how we compute precision of query outputs. The features of the metadata databases employed in the experiments are discussed in Section 6.2. Both groups of queries which are run over the metadata databases and inverted indices are given in Section 6.3. Finally, we present the experimental results in Section 6.4.

6.1 Performance Evaluation Criteria

When a user obtains some information (in the form of documents) from an information retrieval system, he/she will evaluate them according to his/her information need. The documents retrieved by the system will seldom match exactly the user's information need. Some documents may not be useful to the user; others may be helpful in some way. This evaluation measure corresponds to the *relevance* of retrieved documents [9]. If the user judges that a particular document is what he/she is looking for, or it helps him/her in some way with his/her problem, the document is called *relevant* to the information need.

To evaluate performance of information retrieval systems, several criteria such as *coverage*, *time lag*, *recall*, *precision*, *presentation*, *user effort*, etc. have been used [24]. Of these criteria, recall and precision have most frequently been applied [37]. Assuming that the user of an information retrieval system could specify

every document retrieved as relevant or not to his/her particular query or need, *recall* and *precision* are defined as follows:

$$Recall = \frac{Number\ of\ relevant\ documents\ retrieved}{Total\ number\ of\ relevant\ documents\ in\ the\ collection} \tag{6.1}$$

$$Precision = \frac{Number\ of\ relevant\ documents\ retrieved}{Total\ number\ of\ documents\ retrieved} \tag{6.2}$$

However, the measurement of recall on the Web is problematic; due to the extremely dynamic character of the Web, its very high changeability, and its huge size, it is not possible to determine the total number of relevant documents on the Web for a user's query. Therefore, precision is the major performance measure to evaluate Web search systems [37].

As precision of an information retrieval system is the ratio of the number of relevant documents retrieved to the total number of documents retrieved by the system, this value highly depends on relevance judgements of the users. In [37], a model of calculating relevance for hyper-linked document systems is described. According to the model, documents which are directly relevant are assigned a score called the *base relevance*, denoted by *br*. The overall relevance of a document is denoted by *r*, and for directly relevant documents *r*=*br*. Documents which are not directly relevant are assigned a relevance score by incorporating base relevance and inter-document link structure. For those documents the overall relevance r is calculated as

$$r = \begin{cases} br - dr & \text{if } br > dr \\ 0 & \text{otherwise} \end{cases} \tag{6.3}$$

where *dr* is the distance component and calculated as a number of links that need to be traversed to reach to a directly relevant document.

Samalpasis et al. [76] have suggested a simpler version of this relevance model for the Web environment, and their proposal is highly accepted since the model is computationally simple and sufficient in practice [37]. This model assumes that

users tend to examine the first 10 or 20 documents in the result, and they do not usually follow links in very deep levels. Thus, only local link information needs to be incorporated into the relevance judgements. In line with these assumptions, relevance scores are assigned to Web documents as presented in Table 6.1.

Table 6.1: Relevance score values

Relevance Score	**Description**
3	The most relevant document
2	Partly relevant or contains a link to a page with score of 3
1	Somewhat relevant (i.e., short mention of a topic or terms appear on a page) or contains a link to a page ranked 2
0	Not relevant, no query terms are found

By using the relevance measure described in Table 6.1, the inter-document link structure is also incorporated into precision calculations. In [37], four different precision measures are described:

$$Full\,precision = \frac{\sum_{i=1}^{minFnHits} score_i}{minFnHits \times maxHitScore} \tag{6.4}$$

$$Best\,precision = \frac{count\_of_{i=1}^{minFnHits}(score_i = 3)}{minFnHits} \tag{6.5}$$

$$Useful\,precision = \frac{count\_of_{i=1}^{minFnHits}(score_i \geq 2)}{minFnHits} \tag{6.6}$$

$$Objective\,precision = \frac{count\_of_{i=1}^{minFnHits}(score_i > 0)}{minFnHits} \tag{6.7}$$

where $score_i$ is the score assigned to document i; n is the number of measured hits ($n = 10$ for this experiment); *minFnHits* is the minimum of n and *hitsReturned*; *hitsReturned* is the total number of hits returned; and *maxHitScore* is

the maximum relevance score that can be assigned to one hit ($maxHitScore = 3$ in our experiment).

The only difference between the above precision measures is the usage of relevance scores. *Full precision* takes fully into account the subjective score assigned to each hit, and assumes that relevance scores are additive (e.g., 2 documents with scores 3 are equivalent to 6 documents with scores 1). *Best precision*, on the other hand, considers only the most relevant hits. It maps relevance scores to a binary measure. In the *useful precision*, only the most relevant hits, and hits containing links to the most relevant documents are taken. *Objective precision*, as its name implies, is an objective measure since it does not rely on human relevance judgments. It is based on presence or absence of query terms. In this experiment, we employed all these four precision measures along with the relevance score assignment described in Table 6.1.

6.2 Metadata Databases Employed in the Experiment

As we mention at the beginning of this chapter, we employed two expert advice databases, namely Stephen King, and DBLP Bibliography metadata databases, in the performance evaluation experiments. The following two subsections describe these two metadata databases, respectively.

6.2.1 Stephen King Metadata Database

Stephen King metadata database contains expert advice about the famous horror novelist Stephen King and his publications. A domain expert manually created the Stephen King metadata database by browsing hundreds of Web sites about him. The database includes all the novels and books written by Stephen King, their publishers and publication years, and movies and tv-films based on Stephen King novels and books. The metadata database consists of 157 topic instances of

type “horror novelist”, “novel”, “book”, “publisher”, “year”, “movie”, “tv-film”, etc., 304 metalink instances of types *WrittenBy*, *PublisherOf*, *PublicationYear*, *BasedOn*, *Prerequisite*, and *RelatedTo*, and 280 topic sources (i.e., URLs of information resources). Metalinks included in the expert advice database have the following signatures:

WrittenBy: Author-id → *WrittenBy* Novel-id

PublisherOf: Novel-id → *PublisherOf* Publisher-id

PublicationYear: Novel-id → *PublicationYear* Year-id

BasedOn: Novel-id → *BasedOn* Movie-id

BasedOn: Novel-id → *BasedOn* Tvfilm-id

Prerequisite: Novel-id → *Prerequisite* Novel-id

RelatedTo: Novel-id → *RelatedTo* Novel-id

As an example, the metalink instance “Night Shift → *BasedOn* Cat’s Eye” of metalink type *BasedOn* is read as “the movie Cat’s Eye is based on the novel Night Shift”.

Topic sources may also have types (roles) such as “bibliography”, “biography”, “link”, “news”, “commercial”, “picture”, etc., and a topic instance may have more than one source, a Web page may be source for more than one topic instance, and a Web document can be topic source of a topic instance with different source types. For example, the information resources pointed by the URL www.stephenking.com can be topic sources for the topic “Stephen King” with roles “biography” and “bibliography”, and it can also be a topic source for the topic “Richard Bachman” of type “bibliography”. Each source is assigned a real-valued importance in the range [0, 1] by the domain expert such that, the value 1 indicates that the source has the highest importance, and the value 0 means that the source is not important.

6.2.2 DBLP Bibliography Metadata Database

DBLP Bibliography server [51] includes bibliographic information about more than 225,000 computer science publications (e.g., conference and journal papers, books, master and PhD theses, etc.). The DBLP Bibliography metadata database contains expert advice on the publications located at the DBLP Bibliography site. The metadata database contains 380,823 topic instances of topic types "PaperName", "AuthorName", "JournalConference-and-Year", "PublicationDate", and "JournalConferenceOrg", 224,066 distinct topic source instances, 4,902,671 metalink instances of types *InPublicationDate*, *PublicationDateOf*, *JourConfOf*, *JourConfPapers*, *AuthoredBy*, *AuthorOf*, *PrerequisitePapers*, and *RelatedToPapers* having the following signatures:

InPublicationDate: PublicationDate → *InPublicationDate* PaperName

PublicationDateOf: PaperName → *PublicationDateOf* PublicationDate

JourConfOf: PaperName → *JourConfOf* JournalConference-and-Year

JourConfPapers: JournalConference-and-Year → *JourConfPapers* PaperName

AuthoredBy: AuthorName → *AuthoredBy* PaperName

AuthorOf: PaperName → *AuthorOf* AuthorName

PrerequisitePapers: PaperName → *PrerequisitePapers* PaperName

RelatedToPapers: PaperName → *RelatedToPapers* PaperName

We generated the DBLP Bibliography metadata database from the DBLP Bibliography data which is a 90 megabyte sized XML document containing bibliographic entries. We obtained the DBLP Bibliography data from the DBLP Bibliography site, and constructed the expert advice by implementing a collection of Perl and C codes. We exploited the DTD of the DBLP Bibliography data as described in Chapter 3 to generate the expert advice.

Each topic instance of type PaperName is assigned a real-valued importance score in the range [0,1] by considering the impact factor of the journal or conference proceedings that the paper is published in. Table 6.2 presents the scale for the importance score values assigned to PaperNames. For instance, we assign importance score of 1 to a paper which is published in a journal/conference proceeding having top 10% impact factor. We obtained the impact factor values for journals and some conference proceedings from the Web site of CiteSeer [23] search engine which is developed for searching Computer Science publications. Papers published in a journal or conference proceeding whose impact factor is not listed by the CiteSeer, were assigned an importance score of 0.3. Each topic instance of types JournalConference-and-Year and JournalConferenceOrg are assigned importance in the same way as PaperName topic instances. The importance score for each of the metalink instances of type *PrerequisitePapers* and *RelatedToPapers* is the cosine similarity of the paper names that are associated with the metalink type. All other topic and metalink instances are assigned an importance score of 1.

Table 6.2: Importance score scales for publications.

Importance Score	**Impact Factor (top %)**
1	10%
0.9	20%
0.8	30%
0.7	40%
0.6	50%
0.5	60%
0.4	70%
0.3	80%
0.2	90%
0.1	100%

6.3 Queries

For the performance evaluation of our Web querying system, we designed two groups of queries, and run these queries over both metadata databases. Then we

computed the precision of the outputs, as we described in Section 6.1. We also measured the running time of the queries.

6.3.1 Queries Involving SVA Operators

In this thesis, as we propose SQL extensions that can be used for querying the Web, we designed and implemented the below queries each of which serves to satisfy information need of a typical user of the DBLP Bibliography server. Each one of the queries discussed in this section includes at least one SVA operator.

As a researcher, frequently we need to get a list of research papers on a particular topic, and also have the papers in the list be ordered according to their importance. For instance, it will be more useful to us if a research paper that has the most relevance to the searched topic, attracted the highest number of citations, or was published in a highly respected journal or conference proceeding, is located at the top of the result list. The query below can satisfy this kind of user request.

DBLP Query 1: (*SVA Selection*) Using the advice at www.DBLPandAnthology.com/advice, find the names and URLs of 25 highest topic importance ranked papers having most similar titles to the string "XML data and their query languages". Employ a product based topic importance propagation function.

select T.TName, S.URL
using advice at www.DBLPandAnthology.com/advice **as database** DB
from DB.Topics T, DB.TSRef S
where T.TType="PaperName" **and**
T.TName ≅ "XML data and their query languages" **and**
T.TId **in** S.TId
propagate importance as product **function of** T
stop after 25 **most important**

This query finds top 25 topic importance ranked journal or conference papers

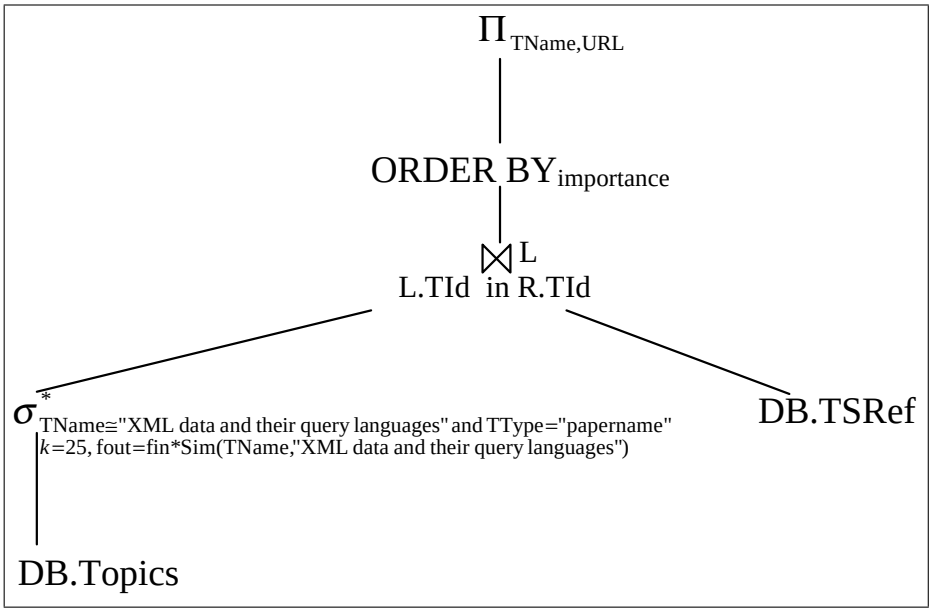

Figure 6.1: Query tree for query1

having titles most similar to the string "XML data and their query languages". The query answers this request by involving one SVA selection operator. The clause T.TName ≅ "XML data and their query languages" states that k topics are selected such that they have the highest name similarity to the string "XML data and their query languages" and their modified importance scores are among the highest k scores over all topics that satisfy the selection condition. The value for k is specified in the stop after clause. The propagate importance clause specifies the function that is used during the score modification of the input topics of the query. Thus, in this query, the score of each paper, which is assigned during the metadata generation step by considering the impact factor of the journal/conference proceeding in which the paper is published, is multiplied by the similarity of the paper title with the string "XML data and their query languages". The SVA selection operator selects top 25 topics having the highest modified scores. After finding the topics that satisfy the selection condition, the sources (Web addresses) for these papers are found by simply joining the output of the SVA selection operator with the TSRef table. The query tree for this query is given in Figure 6.1.

In some other cases, we may want to see a list of research papers which are

similar to some set of papers that we are interested in. SVA directional join operator is suitable for such kind of queries.

DBLP Query 2: (*SVA Directional Join*) Using the advice at www.DBLP-andAnthology.com/advice, for each paper presented in VLDB 2001 conference, find 5 most similar journal or conference papers and their sources. Employ a product based topic importance propagation function.

select T1.TName, S1.URL, T2.TName, T2.URL
using advice at www.DBLPandAnthology.com/advice **as database** DB
from DB.Topics T1, DB.Topics T2, DB.Topics T3,
DB.TSRef S1, DB.TSRef S2, DB.JourConfOf M
where T1.TType = "PaperName" **and** T2.TType = "PaperName" **and**
T3.TType= "JournalConference-and-Year" **and**
T3.TName= "VLDB2001" **and**
T3.Tid = M.Cons-Id **and** T1.Tid = M.Ant-Id **and**
T1.TName $\cong_{(dir,k=5)}$ T2.TName **and**
T1.TId **in** S1.TId **and** T2.TId **in** S2.TId
propagate importance as product **function of** T1, T2

In this query, first, the set of papers that are presented in VLDB 2001 conference are found using the Topics and JourConfOf relations of the DBLP Bibliography metadata. After finding the required set of papers, the SVA directional join operator allows us to find 5 most similar papers for each paper presented in VLDB 2001. The SVA directional join operator is denoted by T1.TName $\cong_{(dir,k=5)}$ T2.TName, and it states that for each TName attribute value for relation T1, find 5 topics (tuples) from relation T2 having most similar topic names to T1.TName. Here, T1 relation contains all the papers presented in VLDB 2001, and T2 relation contains all the journal and conference papers. As product function over T1 and T2 relations is specified as importance propagation function, similarity score between two topic names is calculated as the product of the scores of the topics with the similarity measure between their topic names. The Web addresses for the paper pairs are found by just joining the T1 and T2 relations with the TSRef table. The query tree for Query 2 is provided in Figure 6.2.

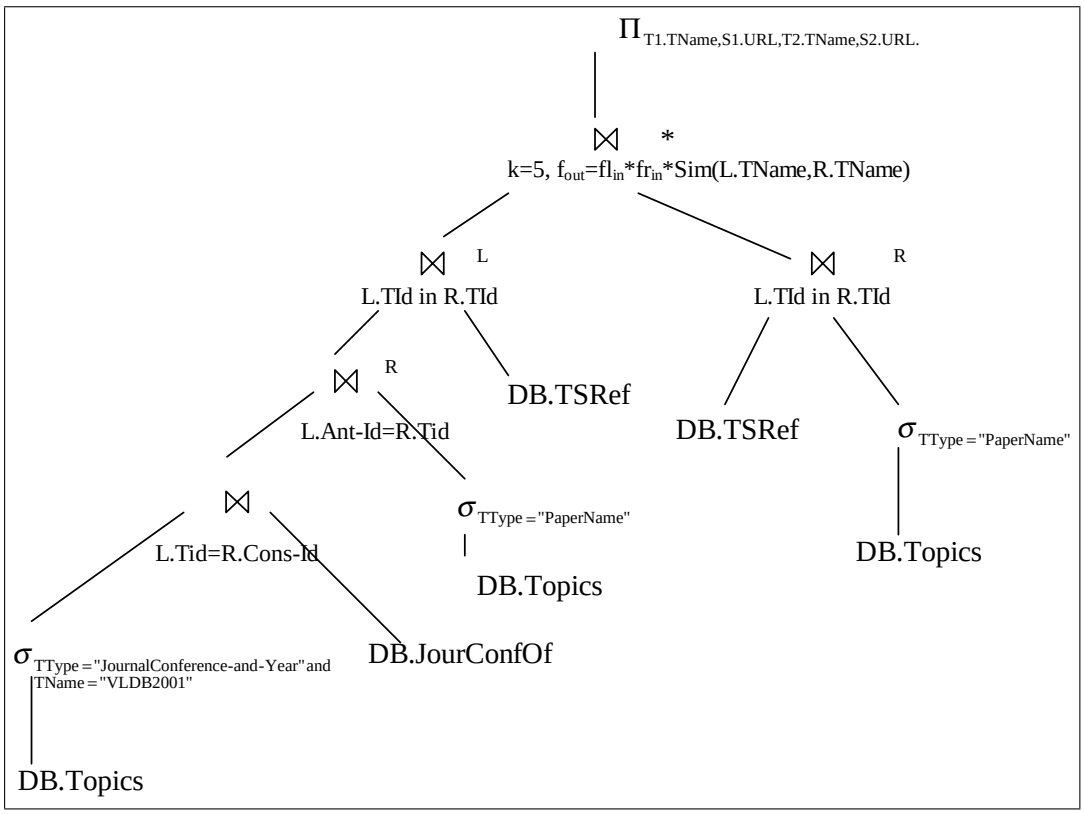

Figure 6.2: Query tree for query 2

We present another useful query below which allows us to see the list of publications to be examined before starting to study a particular topic/paper that we are not familiar with. Any typical scientific paper search service cannot answer this kind of query, but it can be answered by employing our SVA topic closure operator.

DBLP Query 3: (*SVA Topic Closure*) Using the advice at www.DBLPandAnthology.com/advice, find the titles and URLs of 20 highest importance-valued papers such that the selected papers are prerequisite papers to the paper having the title "Information Retrieval on the World Wide Web".

select T2.TName, S2.URL
using advice at www.DBLPandAnthology.com/advice **as database** DB
from DB.Topics T, DB.Topics T1, DB.Topics T2, DB.PrerequisitePapers M, DB.TSRef S2
where T1.TName= "Information Retrieval on the World Wide Web" **and**
T1.TType= "PaperName" **and**
T2.TId **in** *PrerequisitePapers**(T1.TId, T, M) **and**
T2.TId **in** S2.TId
topic closure importance computation as
product **function within a path and as**
max **function among multiple paths**
stop after 20 **most important**

The query includes one topic closure operator which is represented by T2.TId **in** *PrerequisitePapers**(T1.TId, T, M) clause. In the topic closure clause, T1.TId represents the topic-id of the paper having the topic name "Information Retrieval on the World Wide Web", topics represented by T2 are the ones that are reached by following *PrerequisitePapers* metalink paths, specified in table M, originating from the topic T1. During the closure operation, for each new topic that is reached by following the *PrerequisitePapers* metalink paths, importance score of that topic is modified by using the functions specified in the topic closure importance computation clause.

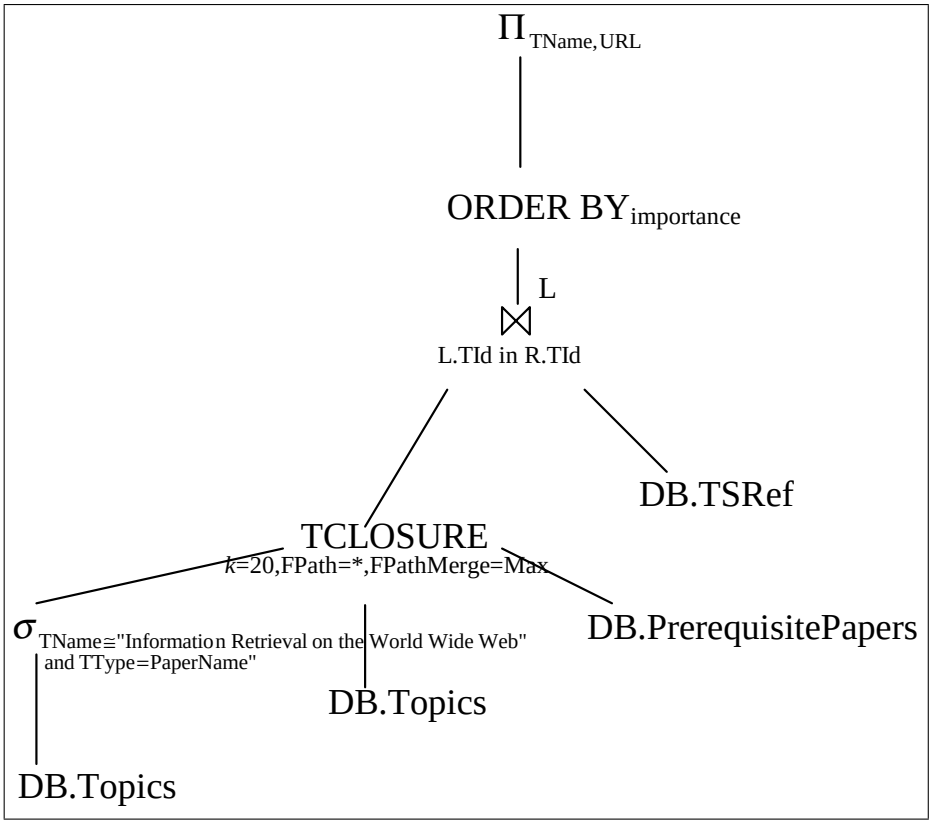

Figure 6.3: Query tree for query 3

Our SQL extensions also allow users to include their preferences and knowledge level into the queries. The fourth query below is an example for personalized metadata-based Web searching.

DBLP Query 4: (*User Profile*) Using the advice at www.DBLPandAnthology.com/advice, and the user profile at www.DBLPandAnthology.com/user, find the titles and URLs of 10 highest importance-valued papers such that the selected papers are related to the paper having the title "Searching for Multimedia on the World Wide Web".

select T2.TName, S2.URL
using advice at www.DBLPandAnthology.com/advice **as database** DB
using profile at www.DBLPandAnthology.com/user **as database** U
from DB.Topics T, DB.Topics T1, DB.Topics T2, DB.RelatedToPapers M, DB.TSRef S2
where T1.TName= "Searching for Multimedia on the World Wide Web" **and**
T1.TType= "PaperName" **and**
T2.TId **in** *RelatedToPapers**(T1.TId, T, M) **and**
T2.TId **in** S2.TId
topic closure importance computation as
product **function within a path and as**
max **function among multiple paths**
stop after 10 **most important**

This query is very similar to the previous query, however in this query the user employs his/her preferences through **using profile at** clause. Assuming that, the user only wants to see the list of papers having the string "multimedia" in their names, then only these topics are considered in the topic closure operator. Thus, employing user profiles decreases the search space and increases the precision of the output, as we show in Section 6.4. The logical query tree for this query is presented in Figure 6.4.

DBLP Query 5: Using the advice at www.DBLPandAnthology.com/advice, find the titles and URLs of 10 highest importance-valued papers such that the

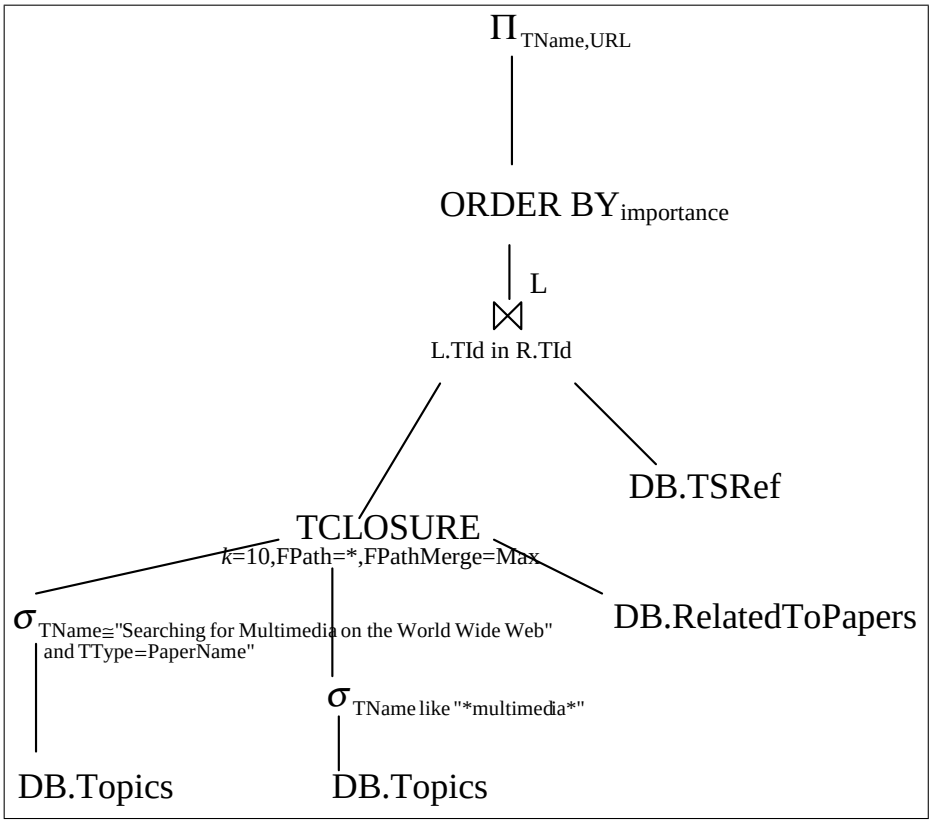

Figure 6.4: Query tree for query 4

selected papers are related papers to the paper having the title "Searching for Multimedia on the World Wide Web".

select T2.TName, S2.URL
using advice at www.DBLPandAnthology.com/advice **as database** DB
from DB.Topics T, DB.Topics T1, DB.Topics T2, DB.RelatedToPapers M, DB.TSRef S2
where T1.TName= "Searching for Multimedia on the World Wide Web" **and**
T1.TType= "PaperName" **and**
T2.TId **in** *RelatedToPapers**(T1.TId, T, M) **and**
T2.TId **in** S2.TId
topic closure importance computation as
product **function within a path and as**
max **function among multiple paths**
stop after 10 **most important**

This query is the same as the previous query except that, in this query no user profile (i.e., user preferences and user knowledge) is employed during the query processing. We designed this query to compare the results obtained with the previous query, and to show the effect of user profile on query processing and in the query outputs.

For the Stephen King matadata database, we also designed 5 queries, which are equivalent to the above queries. By equivalence we mean that, the queries which run over the Stephen King metadata database have the same number of SVA operators with the DBLP Bibliography queries. Also, the query trees and the processing order of operators are the same for both Stephen King and DBLP queries. Only the names of the relations, and the string constant values are different in the query trees. The queries for the Stephen King metadata database are the following:

S.King Query 1: (*SVA Selection*) Using the advice at www.Stephen-King.com/advice, find the names and URLs of 25 highest topic importance ranked novels having titles most similar to the string "dark tower". Return only the URLs

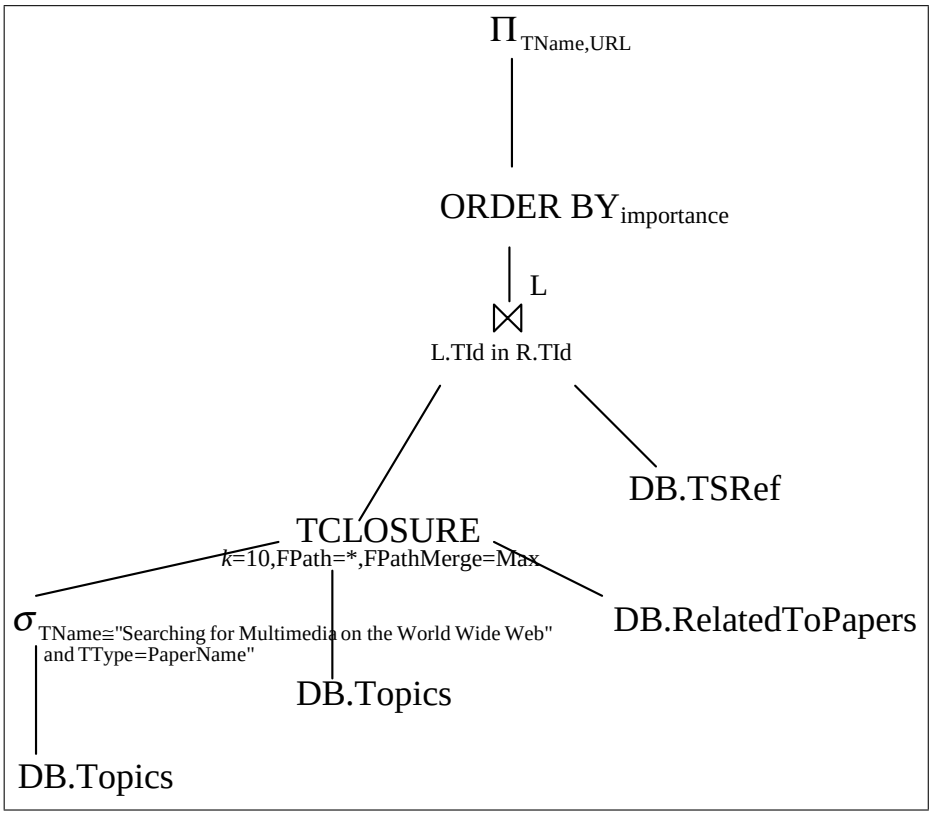

Figure 6.5: Query tree for query 5

of the topic sources of type "bibliography", and employ a product based topic importance propagation function.

S.King Query 2: (*SVA Directional Join*) Using the advice at www.StephenKing.com/advice, for each novel written by Stephen King, find 5 movies or tv-films having titles most similar to the title of the novel. Return their sources of type "summary", and employ a product based topic importance propagation function.

S.King Query 3: (*SVA Topic Closure*) Using the advice at www.StephenKing.com/advice, find the titles and URLs of 20 highest importance-valued novels such that the selected novels are prerequisite to the novel having the title "Wizard & Glass". Return their "description" type of sources.

S.King Query 4: (*User Profile*) Using the advice at www.StephenKing.com/advice, and the user profile at www.StephenKing.com/user, find the titles and URLs of 10 highest importance-valued novels such that the selected novels are related to the novel having the title "Night Journey". Return their "description" type of sources.

S.King Query 5: Using the advice at www.StephenKing.com/advice, find the titles and URLs of 10 highest importance-valued novels such that the selected novels are related to the novel having the title "Night Journey". Return their "description" type of sources.

As these queries are almost the same as the queries over the DBLP Bibliography metadata database, we do not give the extended SQL statements and query trees in this chapter, however, they are presented in Appendix A.1.

We run the above queries over the DBLP Bibliography and Stephen King metadata databases, then we computed and compared the precision of the outputs. The aim of this experiment is to justify the efficiency of the search over metadata databases performed by employing our specialized operators. This experiment also allowed us to compare the precision of the results obtained through semi-automatically generated metadata against the precision values obtained with

manually generated metadata. The results of this experiment are presented in Section 6.4.

6.3.2 Queries without Any SVA Operators

In this part of the experiment, we designed and run the queries given below, which do not include any specialized SVA operator and can also be formulated as keyword queries, over the metadata databases and the inverted indices. Then, the first 10 results returned for each query were examined for the precision computations. The queries made use of in the experiment are:

S. King Query 1: Find all novels written by Stephen King.

S. King Query 2: Find reviews for the novel "Carrie".

S. King Query 3: Find biography of Stephen King.

S. King Query 4: Find the list of Stephen King books published in year 1999.

S. King Query 5: Find commercial sites for the novel "Dark Half".

S. King Query 6: Find all novels of Stephen King, which are not published by "Viking".

S. King Query 7: Find the latest work of Stephen King.

S. King Query 8: Find the description or summaries of all movies and tv-films based on the novel "Dark Half".

S. King Query 9: Find the description or summaries of all movies and tv-films based on the novel "Night Shift".

S. King Query 10: Find the publication year and the publisher of the book "Dead Zone".

S. King Query 11: Find the summary and characters of the book "Dreamcatcher".

All these queries were formulated as both extended SQL and keyword queries. As an example, the SQL statement for Query 1 is the following:

select T.TName, S.URL
using advice at www.StephenKing.com/advice **as database** DB
from DB.Topics T, DB.Topics T1, DB.WrittenBy M, DB.TSRef S
where T.TId=M.Cons-Id **and**
T1.TName="Stephen King" **and** T1.TType="Author" **and**
M.Ant-Id=T1.Tid **and** T.TId **in** S.TId
order by S.S-advice **desc**

The same query is also formulated as "Stephen" AND "King" AND "novel", and as "Stephen King novel" for boolean and ranked querying, respectively. The extended SQL statements for all the queries which run over the Stephen King metadata database are provided in Appendix A.2.

The queries that are used in this experiment are chosen such that, the first 4 queries are simple general search queries, and they can easily be formulated and expected to return relevant hits by any search system. The last 7 queries, on the other hand, are more complicated, and more specific information is sought for. These queries are chosen to confirm the querying power of metadata-based search over keyword-based search systems.

The equivalent queries[6] that were run over the DBLP Bibliography metadata database are:

DBLP Query 1: Find all papers written by J. D. Ullman.

DBLP Query 2: Find the document(s) containing the full text of the paper "Mining Sequential Patterns".

DBLP Query 3: Find the homepage of J. D. Ullman.

[6]in terms of query trees and the processing order of operators

DBLP Query 4: Find the list of J. D. Ullman papers published/presented in year 2000.

DBLP Query 5: Find sources for the paper "Access Methods for Text".

DBLP Query 6: Find all papers of J. D. Ullman, which are not presented in any VLDB Conference.

DBLP Query 7: Find the latest published paper of J. D. Ullman.

DBLP Query 8: Find the list of all papers which are directly related to the paper "Access Methods for Text".

DBLP Query 9: Find the list of all papers which are directly related to the paper "Mining Sequential Patterns".

DBLP Query 10: Find the year and the journal/conference name in which the paper "Access Methods for Text" is published/presented.

DBLP Query 11: Find the bibliographic information of the paper "Mining Sequential Patterns".

6.4 Experimental Results

The full, best, useful, and objective precision of the outputs generated by the first group of queries are presented in Figure 6.6 and Figure 6.7. As shown in the Figure 6.7, the objective precision value for all queries is 1 for both Stephen King and DBLP Bibliography metadata databases, meaning that all hits returned by querying the two expert advice systems are relevant (i.e., having relevance scores greater than 0), and no bad hits are returned. The main reason for having 1 as objective precision is that, all of the documents covered by the metadata databases are relevant documents (i.e., all documents are about "Stephen King"), so they are scored more than 0 by the query posers. Thus in this experiment, objective precision is not a sufficient measure. On the other hand, the full, best, and useful precision values may not be objective since they highly depend on

relevance judgments. Nevertheless, these measures are useful in the sense that, they show the ability of a search system to retrieve highly relevant documents (i.e., having score ≥ 2).

According to Figures 6.6 and 6.7, for queries 2 and 3, the full, best, and useful precision values are quite close to each other for the two metadata databases. For the other queries, precision of the DBLP Bibliography metadata database is less than the Stephen King metadata database because of two main reasons: firstly, the DBLP Bibliography metadata database covers huge number of information resources (more than 225,000 publications), secondly, the metadata objects (i.e., topic, metalink, source instances) of the DBLP Bibliography metadata database is generated automatically by considering only the bibliographic information of the publications. The Stephen King metadata database, on the other hand, is created by a human expert by considering the full text of the Web documents. If we could have covered the full text documents of all the publications located at the DBLP Bibliography site, the metadata database would yield more precise results. Nevertheless, the precision of the DBLP Bibliography metadata database is satisfactory as it achieves 80% full precision for query 1, approximately 90% full precision for queries 2,3, and 4. The precision values for query 5 is lower with respect to the other queries for both metadata databases, due to the fact that the metalink type that is involved in this query has too many instances, and some of these instances were not considered as "relavant" by the query poser. When the user preferences are taken into account during the query processing (i.e., query 4), the outputs generated for the same query are assigned higher scores by the user.

During the processing of the same queries involving SVA operators, we also measured the running time of the queries. All the queries were run on a PC having Pentium III 450Mhz CPU, and 320Mb of RAM, and we assumed that all the metadata objects are read from disk, and kept in the memory until the end of the query processing. Table 6.3 presents the running time for the queries over both metadata databases. Processing time of the queries run over the DBLP Bibliography metadata database is higher since this database contains huge number of topic and metalink instances as compared to the Stephen King metadata

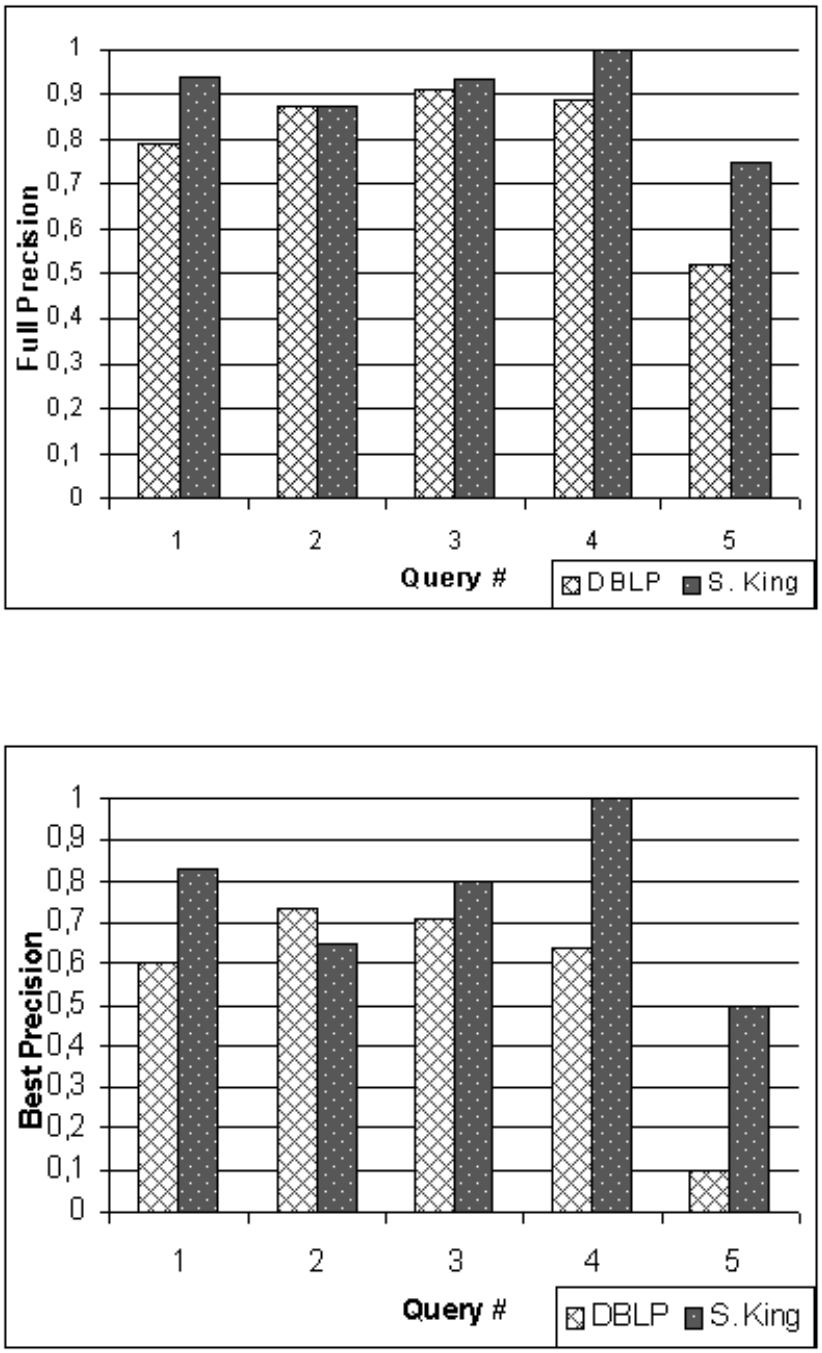

Figure 6.6: Full and best precision of the outputs for queries involving SVA operators.

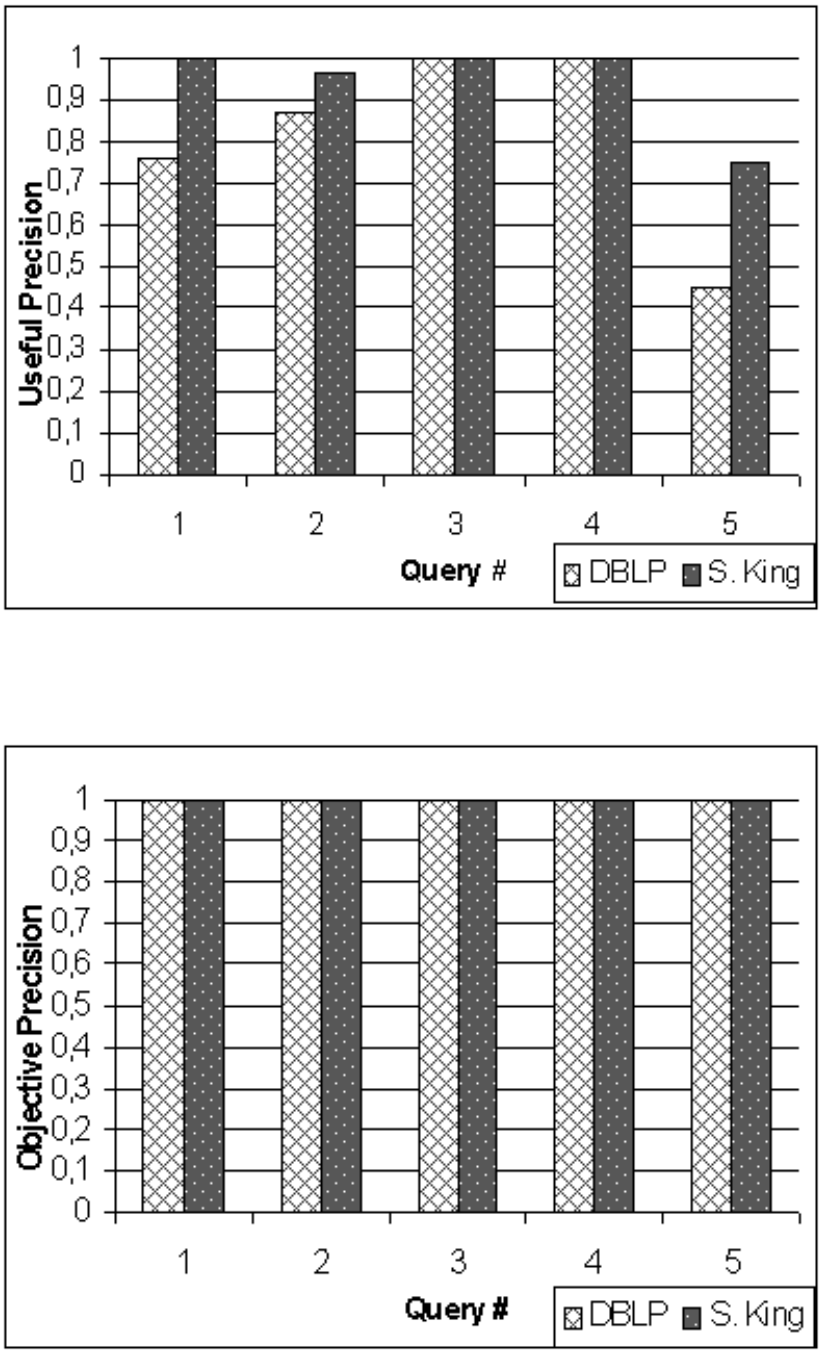

Figure 6.7: Useful and objective precision of the outputs for queries involving SVA operators.

database. Queries involving SVA directional join operator (e.g., query 2) requires more time than the queries including SVA selection (e.g., query 1), as the directional join operation involves much more similarity computations than the SVA selection operation. Both queries 3 and 5 include one topic closure operator, however the metalink type employed in query 5 has many more instances than the one used in query 3, hence query 5 runs slower. Queries 4 and 5 are actually the same except that query 4 includes user preferences during the processing of the topic closure operator. Thus, in query 4, deeper levels are searched during the closure operator to reach topic instances that satisfy user preferences. On the other hand, closure operation is pruned for some topic instances which are not required by the user.

Table 6.3: Running time for the queries involving SVA operators (in seconds)

Query #	**Time for DBLP Database**	**Time for S. King Database**
1	32	<1
2	332	1
3	68	<1
4	738	1
5	134	<1

According to Figures 6.6, 6.7 and Table 6.3, metadata-based Web querying by employing the SQL extensions, allows query posers to form useful queries which can not be formulated by any other search techniques, and yields high precision query outputs in a reasonably short amount of time. Employing user preferences during the query processing helps to improve the precision of the output further.

For the second part of the experiment, we created one inverted index for each metadata databases. The inverted index for the Stephen King metadata database is created over full text documents covered by the expert advice. For the DBLP Bibliography database, we built the inverted index over the bibliographic entries of the publications. We did not consider the full text documents of the publications. Our DBLP Bibliography expert advice is also generated over the bibliographic entries only, since obtaining and indexing the full text documents of more than 225,000 publications requires vast amount of work.

Figures 6.8 through 6.11 present the full, best, useful, and objective precision of the outputs generated by the second group of queries which do not involve any SQL extension. As in the case of the first part of the experiment, the objective precision of the outputs for all queries and all search techniques (i.e., metadata based, boolean, and ranked) are equal to 1, because of the fact that, metadata databases and the inverted indexes cover only relevant documents on a particular domain.

As displayed in Figures 6.8 through 6.11, the full, best and useful precision values for boolean and ranked querying hits are not as high as objective precision values, which show that most of the relevant hits returned by these keyword based search systems have scores 2 and 1. As a result of this, useful precision of keyword based querying systems is slightly higher than their full precision; best precision on the other hand, is quite low for these systems. According to the figures, metadata-based querying over both metadata databases beats keyword based search with respect to all the three precision measures. This is due to the fact that both keyword based querying systems would not probably retrieve the highest scored documents in the top 10 hits, because these systems can only return documents that contain the search terms (in any context). As metadata-based search employs expert advices, it can return the most relevant documents in the first 10-result set.

For the Stephen King metadata database (Figures 6.8 and 6.9), only for a few queries (Queries 3, 8, and 10), the best and full precision values are slightly less than 1, implying that only a few number of the documents returned in the result sets have scores less than 3. This result occurred because of the fact that, for these queries, the number of Web documents having score 3 that are included in the Stephen King expert advice repository is less than 10, or in some occasional cases the relevance judgements of query posers may not agree with the importance assignment of the domain experts.

For the DBLP Bibliography metadata database (Figures 6.10 and 6.11), the full and best precision of only two queries (Queries 8 and 9) are almost equal to the ranked querying, and slightly less than the boolean querying. This result shows

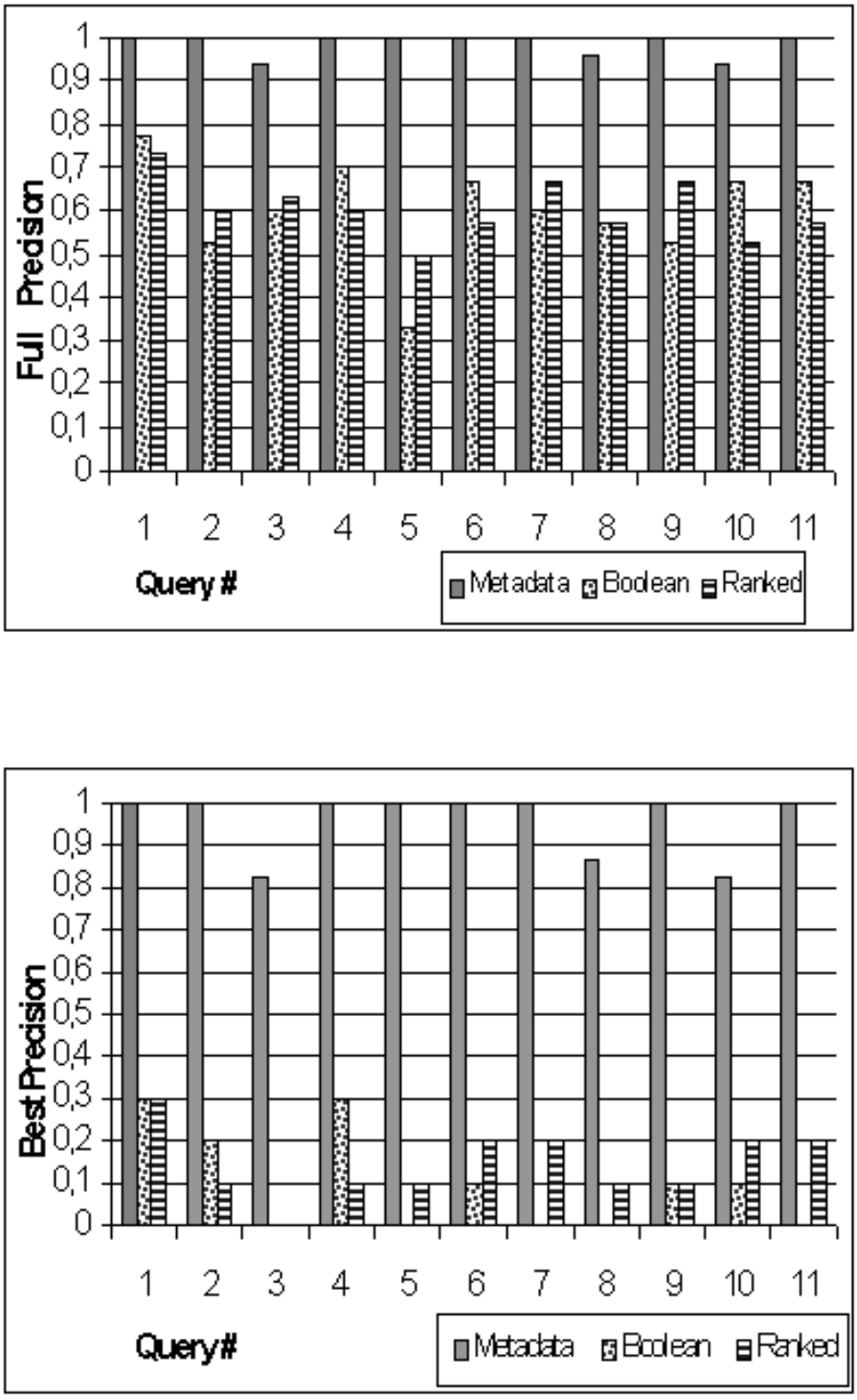

Figure 6.8: Full and best precision of the outputs for queries not involving SVA operators and run over the Stephen King metadata database.

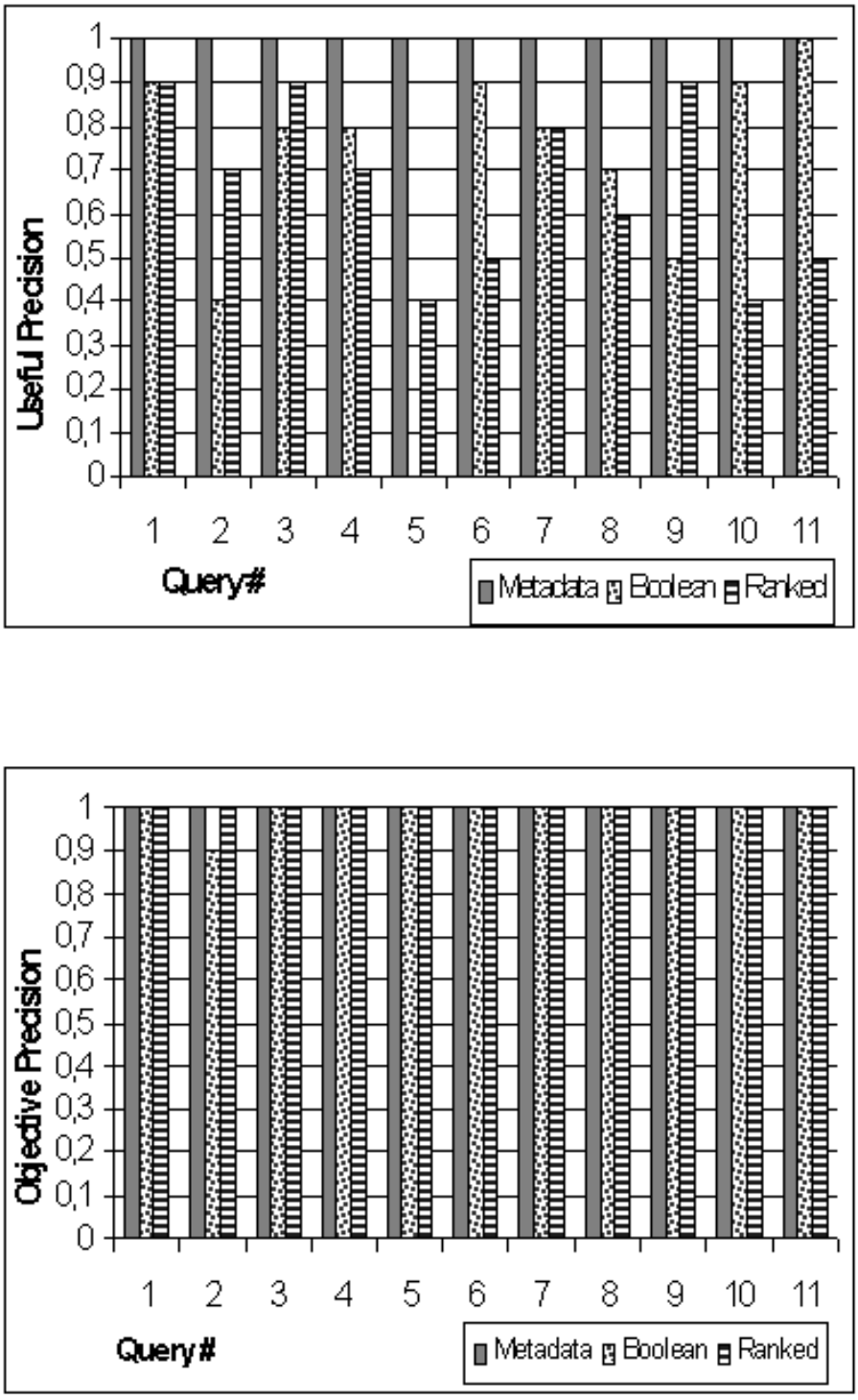

Figure 6.9: Useful and objective precision of the outputs for queries not involving SVA operators and run over the Stephen King metadata database.

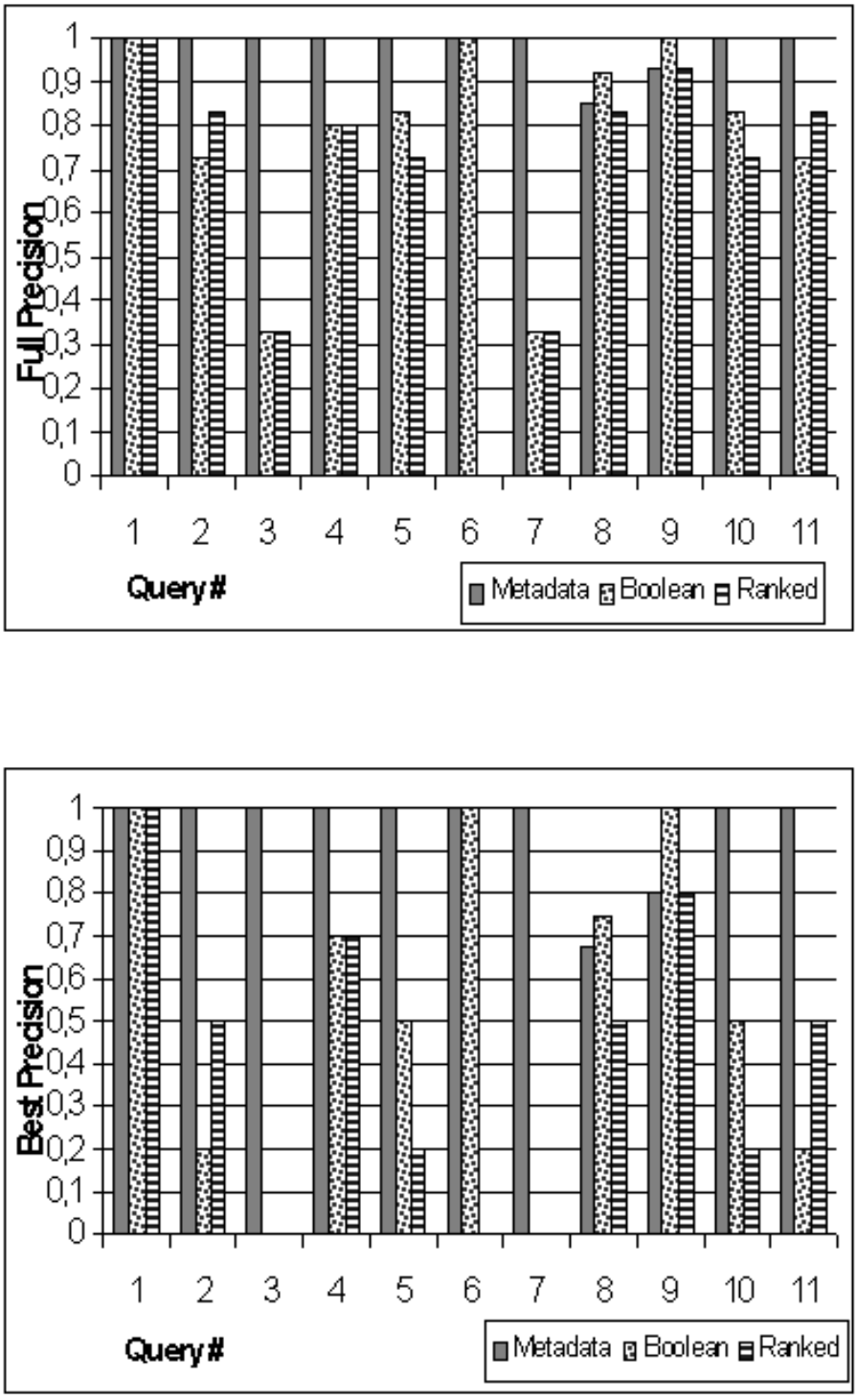

Figure 6.10: Full and best precision of the outputs for queries not involving SVA operators and run over the DBLP Bibliography metadata database.

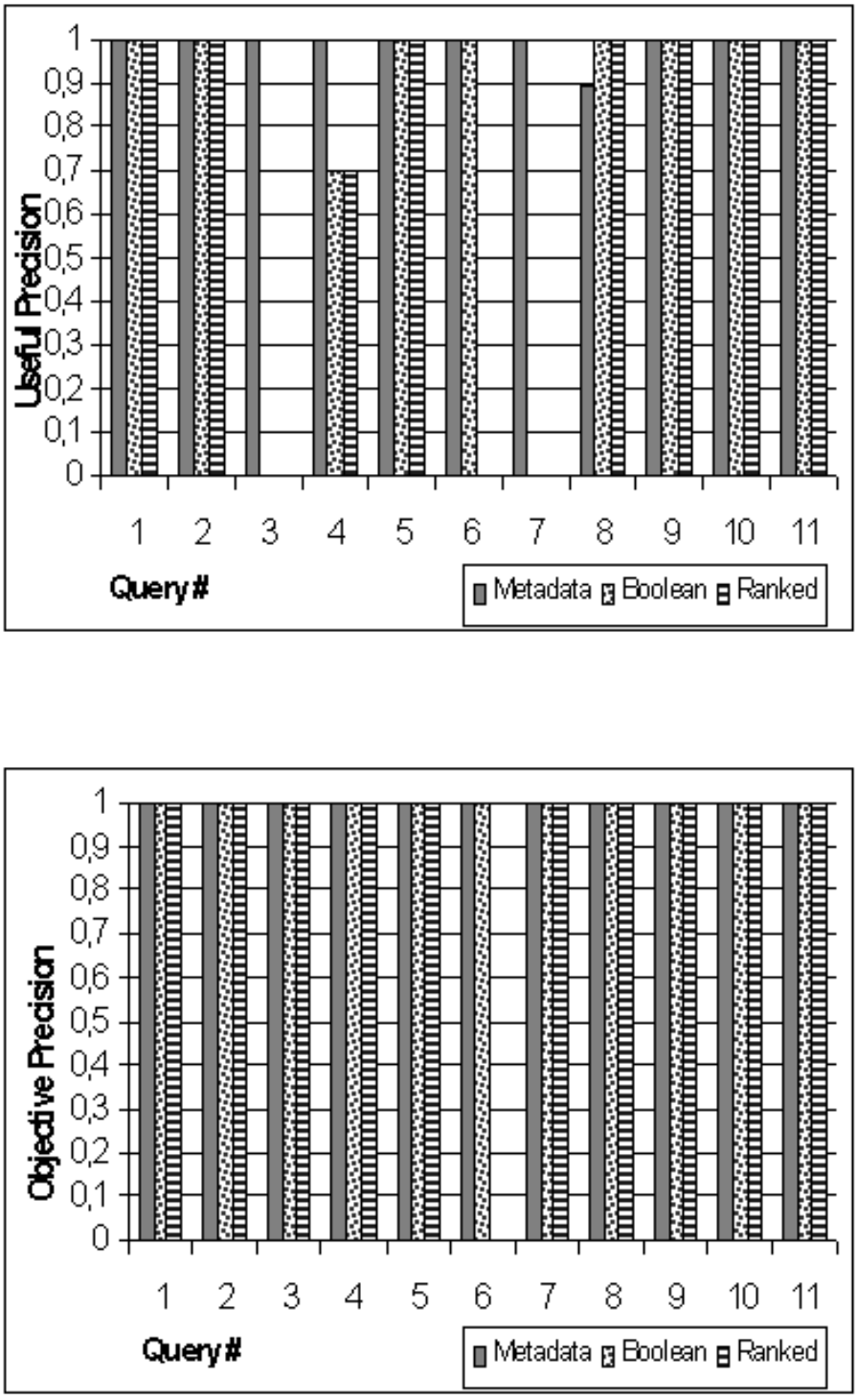

Figure 6.11: Useful and objective precision of the outputs for queries not involving SVA operators and run over the DBLP Bibliography metadata database.

that the rule used to determine the related papers during the metadata generation step does not match exactly with the user request. However, if the query poser could employ user preferences during the query processing, the precision would be higher, as we observed in the first part of the experiment.

When we compare the precision of both expert advices with each other, regardless of being manually or semi-automatically created, metadata databases yield high precision results as compared to keyword based search over an inverted index. However, the gap between the precision of the manually generated metadata-based search and keyword based search is higher.

In the second part of the experiment we also showed that making metadata-based search is favorable for some kind of queries. As an example, in Query 5 for the Stephen King database, the query poser wants to see some buying information about the novel "Dark Half". Metadata-based querying responds to this query by directly listing the information resources having the role "commercial" for the novel "Dark Half". Ranked and boolean querying systems on the other hand, try to find Web documents containing the keywords "Dark Half", "novel", and "commercial", and respond poorly to this query. In Query 7, metadata-based querying directly gives the list of novels/books with the latest publication date and written by Stephen King. However, to find the latest works of him, a keyword search based system user has to browse tens of Web documents returned by the system. Thus, in this experiment, we also observed that metadata-based search performs better than keyword-based search when specific information is queried, and queries involve relationships among query terms.

Chapter 7

Conclusions and Future Work

In this thesis we have described a metadata-based querying system for information resources on the Web. We have presented a data model named Web information space model, for metadata-based modeling of Web resources. Our Web information space model consists of three main parts: *information resources*, *metadata databases*, and *user profiles*. The information resources include the Web-based information sources to be queried. Metadata databases store metadata about the information resources in terms of topics, relationships among topics (i.e., metalinks), and references to the information resources that they describe. User profiles contain user preferences about expert advices, user knowledge level about topics, and navigational history information about the Web documents that the user visits. In order to make the metadata generation an attainable task, we have assumed that information resources in the Web information space model do not cover the whole Web; rather they are defined within a set of Web resources on a particular domain, which we call subnets.

In our model, metadata and user profiles are stored in a (object) relational database management system. In order to query the Web information resources efficiently, we have enriched the SQL with score management functionality, and new operators like text similarity based selection, text similarity based join, and topic closure. Score management allows ranking of query outputs, and limiting the output size. Text similarity based operators provide text similarity based

querying over the information resources. Topic closure operator supports recursive closure queries that can not be formed by keyword matching based Web search engines. Our SQL extensions also allow query poser to include his/her profile (i.e., preferences and knowledge level) into the queries to facilitate personalized Web search. We have also defined an algebra named as Sideway Value generating Algebra (SVA) to support our SQL extensions, and presented algebraic operators along with their processing algorithms. We have especially focused on the text similarity based SVA directional join operator and proposed a new algorithm for efficient processing of the directional join operator.

Text similarity based directional join is a very useful operator to be employed in a variety of applications, such as the integration of distributed, heterogeneous databases that lack common object identifier; querying a multidatabase system that manages both relational and text databases; and integration and querying of metadata/ontology from multiple resources to facilitate Web querying. In this thesis, we have evaluated the text similarity based join algorithms proposed previously, developed a new algorithm which is more efficient in terms of the number of tuple comparisons and disk accesses made, and incorporated some early termination heuristics from the Information Retrieval domain to achieve further improvement in the performance of our algorithm. We have demonstrated through experimental evaluation that nested loops based join algorithm for the text similarity based directional join operator performs the best in terms of the number of disk accesses required. However, this algorithm compares every tuple pair from the relations to be joined and leads to a huge amount of expensive similarity computations. Inverted index based join algorithms, on the other hand, achieve very small number of similarity computations while requiring large number of disk accesses. We have developed a new nested loop based join algorithm, which employs an in memory inverted index, to implement the similarity based directional text join operation more efficiently. We have observed further performance improvement by applying maximal similarity filter and continue heuristics to our join algorithm.

We have also evaluated performance of metadata-based Web querying against

traditional keyword matching based techniques (i.e., boolean and ranked querying techniques) through experimentation. In the performance evaluation experiments, we have employed two metadata databases (i.e., Stephen King metadata database, and DBLP Bibliography metadata database), and run some test queries over these metadata databases using our extended SQL. Then, we have run the boolean and ranked query versions of the same queries on the inverted indexes that we have created over the Web documents covered by the two metadata databases, and compared the precision of the results returned by the extended SQL, boolean, and ranked queries. We have shown that the precision of the metadata-based querying is higher than that of the boolean and ranked querying. This experiment also allowed us to compare the querying performance of the manually created metadata database (i.e., Stephen King metadata database) against semi-automatically created metadata database (i.e., DBLP Bibliography metadata database). While the performance with both manually and semi-automatically generated metadata databases was observed to be better than that of the keyword based search systems, the overall performance obtained with the manually generated metadata database was at a higher level. In addition to these, we have observed that, employing user preferences during the metadata-based Web querying improves the precision of the results returned.

In this thesis, we have extracted metadata by exploiting DTDs of the XML information resources. Although XML is adopted as a standard for electronic data exchange on the Web, currently most of the Web resources are not XML documents, and efficient metadata extraction tools are required for Web information resources of any media type such as text (e.g., ps, pdf, ascii), image (e.g., jpeg, bitmap), audio, etc.

Although our proposal for metadata-based Web querying includes SQL extensions along with new operators and score management functionality, and we have not dealt with the usage of the extended SQL by naive users. Also, we have implemented only our specialized SQL operators, not a full query processor for the extended SQL. As a future work, a graphical user interface that help to formulate extended SQL queries easily, and a query processor for executing the extended SQL queries can be developed.

Bibliography

[1] S. Abiteboul, P. Buneman, and D. Suciu. *Data on the Web*. Morgan Kaufmann Publishers, San Francisco, 258 p., 2000.

[2] ACM Portal Web Site. Available at http://www.acm.org.

[3] R. Agraval and R. Srikant. Mining Sequential Patterns. *In Proc. ICDE Conference 1995*, pp. 3-14.

[4] R. Agrawal, and E. L. Wimmers. A Framework for Expressing and Combining Preferences. *In Proc. ACM SIGMOD Conference 2000*, pp. 297-306.

[5] İ. S. Altıngövde. Topic-Centric Querying of Web Resources. MS. Thesis, Bilkent University, Deptartment of Computer Engineering, September 2001.

[6] İ. S. Altıngövde, S. A. Özel, Ö. Ulusoy, G. Özsoyoğlu, and Z. M. Özsoyoğlu. Topic-Centric Querying of Web Information Resources. *In Proc. Database and Expert Systems Applications 2001 (DEXA'01)*, pp. 699-711.

[7] G. O. Arocena. WebOQL: Exploiting Document Structure in Web Queries. MS Thesis, Department of Computer Science, University of Toronto, 1997.

[8] G. O. Arocena, A. O. Mendelzon, and G. A. Mihaila. Applications of a Web Query Language, 1997. Available at http://www.cs.toronto.edu/ websql/www-conf/wsql/PAPER267.html.

[9] R. Baeza-Yates, and B. Ribeiro-Neto. *Modern Information Retrieval*. Addison Wesley, 513 p., 1999.

[10] A. A. Barfourosh, H. R. M. Nezhad, M. L. Anderson, and D. Perlis. Information Retrieval on the World Wide Web and Active Logic: A Survey and Problem Definition, 2002. Available at http://citeseer.nj.nec.com/barfourosh02information.html.

[11] R. J. Bayardo, and D. P. Miranker. Processing Queries for First Few Answers. *In Proc. Conference on Information and Knowledge Management 1996 (CIKM'96)*, pp. 45-52.

[12] T. Berners-Lee. Semantic Web Roadmap. W3C draft, 2000. Available at http://www.w3.org/DesignIssues/Semantic.html.

[13] M. Biezunski. Topic Maps at a Glance, 2001. Available at http://www.infoloom.com/tmsample/bie0.htm.

[14] M. Biezunski, M. Bryan, and S. Newcomb (Eds). ISO/IEC 13250, Topic Maps, 1999. Available at http://www.ornl.gov/sgml/sc34/document/0058.htm.

[15] J. Bosak. XML, Java, and the Future of the Web, 1997. Available at http://www.xml.com/xml/pub/w3j/s3.bosak.html.

[16] T. Bray, J. Paoli, C. M. Sperberg-McQueen, and E. Maler. Extensible Markup Language (XML) 1.0 (Second Edition), 2000. Available at http://www.w3.org/TR/REC-xml.

[17] S. Brin and L. Page. The Anatomy of a Large-Scale Hypertextual Web Search Engine, 1998. Available at http://www7.scu.edu.au/programme/-fullpapers/1921/com1921.htm.

[18] M. J. Carey, and D. Kossmann. On Saying "Enough Already!" in SQL. *In Proc. ACM SIGMOD Conference 1997*, pp. 219-230.

[19] M. J. Carey, and D. Kossmann. Reducing the Breaking Distance of an SQL Query Engine. *In Proc. VLDB Conference 1998*, pp. 158-169.

[20] K. C. Chang, and S. Hwang. Minimal Probing: Supporting Expensive Predicates for Top-k Queries. *In Proc. ACM SIGMOD Conference 2002*, pp. 346-357.

[21] S. Chaudhuri, and L. Gravano. Evaluating Top-k Selection Queries. *In Proc. VLDB Conference 1999*, pp. 397-410.

[22] V. Chritophides. Community Webs (C-Webs): Technological Assessment and System Architecture, 2000. Available at http://citeseer.nj.nec.com/christophides00community.html.

[23] CiteSeer Search Engine. Available at http://citeseer.nj.nec.com.

[24] C. W. Cleverdon, J. Mills, and E. M. Keen. An inquiry in testing of information retrieval systems. Cranfileld, U.K.: Aslib Cranfield Research Project, College of Aeronautics, 1966.

[25] W. Cohen. Data Integration Using Similarity Joins and a Word-Based Information Representation Language. *ACM Transactions on Information Systems*, 18(3):288-321, 2000.

[26] A. Deutsch, M. Fernandez, D. Florescu, A. Levy, D. Maier, and D. Suciu. Querying XML Data. *Data Engineering*, 22(3):10-18, 1999.

[27] A. Deutsch, M. Fernandez, and D. Suciu. Storing Semistructured Data with STORED. *In Proc. ACM SIGMOD Conference 1999*, pp. 431-442.

[28] ECDL Workshop on the Semantic Web, Sept. 21, 2000, Lisbon, Portugal.

[29] R. Fagin, A. Lotem, and M. Naor. Optimal Aggregation Algorithms for Middleware. *In Proc. PODS 2001*, pp. 102-113.

[30] M. Fernandez, D. Florescu, A. Levy, and D. Suciu. Declarative Specification of Web Sites with STRUDEL. *The VLDB Journal*, (9):38-55, 2000.

[31] D. Florescu, A. Levy, and A. Mendelzon. Database Techniques for the World Wide Web: A Survey. *ACM SIGMOD Record*, 27(3):59-74, 1998.

[32] H. Folch, and B. Habert. Constructing a Navigable Topic Map by Inductive Semantic Acquisition Methods. *In Proc. Extreme Markup Languages 2000.* Available at www.limsi.fr/Individu/habert/Publications/Fichiers/folch-et-habert00/folch-et-habert00.html.

[33] L. M. Garshol. tolog A Topic Map Query Language, 2001. Available at http://www.ontopia.net/topicmaps/materials/tolog.html.

[34] L. M. Garshol. Topic Maps, RDF, DAML, OIL: A Comparison, 2001. Available at http://www.ontopia.net/topicmaps/materials/tmrdfoildaml.html.

[35] R. Goldman, J. McHugh, and J. Widom. From semistructured Data to XML: Migrating the Lore Data Model and Query Language. *In Proc. WebDB 1999*, pp. 25-30.

[36] L. Gravano, P. G. Ipeirotis, N. Koudas, and D. Srivasta. Text Joins in an RDBMS for Web Data Integration. *In Proc. WWW 2003*, pp. 90-101.

[37] J. Gwidzka, and M. Chignell. Towards Information Retrieval Measures for Evaluation of Web Search Engines, 1999. Available at http://imedia.mie.utoronto.ca/ jacekg/pubs/webIR_eval1_99.pdf.

[38] J. Han, Y. Fu, W. Wang, K. Koperski, and O. Zaiane. DMQL: A Data Mining Query Language for Relational Databases. *In Proc. SIGMOD'96 Workshop on Research Issues in Data Mining and Knowledge Discovery (DMKD'96).*

[39] D. K. Harman and G. Candela. Retrieving Records from a Gigabyte of Text on a Minicomputer Using Statistical Ranking. *Journal of the American Society for Information Science*, 41(8):581-589, 1990.

[40] R. Himmeröder, G. Lausen, B. Ludascher, and C. Schlepphorst. On a Declarative Semantics for Web Queries. *In Proc. International Conference on Deductive and Object Oriented Databases 1997 (DOOD'97)*, pp. 386-398.

[41] I. Horrocks. DAML+OIL: a Description Logic for the Semantic Web. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, 25(1):4-9, 2002.

[42] ISO/IEC 13250 Topic Maps standard, 1999. Available at http://www.y12.doe.gov/sgml/sc34/document/iso13250-2nd-ed-v2.pdf.

[43] M. Kobayashi and K. Takeda. Information Retrieval on the Web. *ACM Computing Surveys*, 32(2):144-173, 2000.

[44] D. Konopnicki, and O. Shmueli. W3QL: A Query System for the World Wide Web. *In Proc. VLDB Conference 1995*, pp. 54-65.

[45] D. Konopnicki, and O. Shmueli. Bringing Database Functionality to the WWW. *In Proc. International Workshop on the Web and Databases 1998*, pp. 49-55.

[46] R. Ksiezyk. Answer is just a question of matching Topic Maps, 2000. Available at http://www.infoloom.com/gcaconfs/WEB/paris2000/S22-03.HTM#s22-03tmql.

[47] M. Kutlutürk. Implementation of a Topic Map Data Model for a Web-based Information Resource, MS. Thesis, Bilkent University, Department of Computer Engineering, Augusut, 2002.

[48] A. H. F. Laender, B. A. Ribeiro-Neto, A. S. da Silva, and J. S. Teixeira. A Brief Survey of Web Data Extraction Tools. *SIGMOD Record*, 31(2):84-93, 2002.

[49] M. S. Lacher, and S. Decker. On the Integration of Topic Maps and RDF Data. *In Proc. Semantic Web Working Symposium 2001.*

[50] L. V. S. Lakshmanan, F. Sadri, and I. N. Subramanian. A Declarative Language for Querying and Restructuring the Web. *In Proc. International Workshop on Research Issues in Data Engineering 1996, RIDE' 96*, pp. 12-21.

[51] M. Ley. DBLP Bibliography, 2001. Available at http://www.informatik.uni-trier.de/~ley/db/.

[52] L. Li. Metadata Extraction: RelatedToPapers and its Use in Querying. MS. Thesis, Case Western Reserve Unversity, Department of Electrical Engineering and Computer Science, August, 2003.

[53] W. S. Li, J. Shim, K. S. Candan, and Y. Hara. WebDB: A Web Query System and its Modeling, Language, and Implementation. *In Proc. IEEE Advances in Digital Libraries 1998*, pp. 216-227.

[54] H. Liefke, and D. Suciu. XMill: an Efficient Compressor for XML Data. In Proc. *ACM SIGMOD Conference 2000*, pp. 153-164.

[55] B. Ludascher, R. Himmeröder, G. Lausen, W. May, and C. Schlepphorst. Managing Semistructured Data with FLORID: A Deductive Object Oriented Perspective. *Information Systems*, 23(8):1-25, 1998.

[56] A. Magkanaraki, G. Karvounarakis, T. T. Anh, V. Christophides, and D. Plexousakis. Ontology Storage and Querying. Technical Report. Foundation for Research and Technology Hellas, Institute of Computer Science, 2002.

[57] G. Mecca, P. Merialdo, and P. Atzeni. Araneus in the Era of XML. *Data Engineering*, 22(3):19-26, 1999.

[58] A. O. Mendelzon, G. A. Mihalia, and T. Milo. Querying the World Wide Web. *International Journal of Digital Libraries*, 1(1):54-67, 1997.

[59] W. Meng, C. Yu, W. Wang, and N Rishe. Performance Analysis of Three Text-Join Algorithms. *IEEE Transactions on Knowledge and Data Engineering*, 10(3):477-492, 1998.

[60] P. Mishra, and M. H. Eich. Join Processing in Relational Databases. *ACM Computing Surveys*, 24(1):63-113, 1992.

[61] Microsoft Developers Network Online Support. Available at http://support.microsoft.com/servicedesks/msdn.

[62] G. A. Mihaila, L. Raschid, and A. Tomasic. Locating and accessing data repositories with WebSemantics. *VLDB Journal*, 11:47-57, 2002.

[63] A. Moffat and J. Zobel. Self Indexing Inverted Files for Fast Text Retrieval. *ACM Transactions on Information Systems*, 14(4):349-379, 1996.

[64] Mondeca home page. Available at http://www.mondeca.com.

[65] G. Moore. Topic Map Technology - the state of the art, 2000. Available at http://www.infoloom.com/gcaconfs/WEB/paris2000/S22-044.htm.

[66] A. Natsev, Y. C. Chang, J. R. Smith, C. S. Li, and J. S. Vitter. Supporting Incremental Join Queries on Ranked Inputs. *In Proc. VLDB Conference 2001*, pp. 281-290.

[67] S. Newcomb, and M. Biezunski. Topic Maps go XML. *In Proc. XML Europe 2000.*

[68] S. A. Özel, İ. S. Altıngövde, Ö Ulusoy, G. Özsoyoğlu, and Z. M. Özsoyoğlu. Metadata-based Modeling of Information Resources on the Web. *Journal of the American Society for Information Science and Technology*, 55(2):97-110, 2004.

[69] G. Özsoyoğlu, A. Al-Hamdani, İ. S. Altıngövde, S. A. Özel, Ö. Ulusoy, and Z. M. Özsoyoğlu. Sideway Value Algebra for Object-Relational Databases. *In Proc. VLDB Conference 2002.*

[70] G. Özsoyoğlu, İ. S. Altıngövde, A. Al-Hamdani, S. A. Özel, Ö. Ulusoy, and Z. M. Özsoyoğlu. Querying Web Metadata: Native Score Management and Text Suport in Databases. submitted to a journal, 2003.

[71] S. Pepper. Euler, Topic Maps, and Revolution. *In Proc. XML Europe 1999.* Available at http://www.infoloom.com/tmsample/pep4.htm.

[72] H. H. Rath, and S. Pepper. Topic Maps at Work. C. F. Goldfarb and P. Prescod (eds): XML Handbook, 2nd edition, Prentice Hall, 1999.

[73] Resource Description Framework (RDF) Model and Syntax Specification. W3C Recommendation, 1999. Available at http://www.w3.org/TR/REC-rdf-syntax/.

[74] A. Sahuguet. Everything You Ever Wanted to Know About DTDs, But Were Afraid to Ask. *In Proc. WebDB 2000*, pp. 69-74.

[75] G. Salton. *Automatic Text Processing.* Addison-Wesley, 1989.

[76] M. Samalpasis, J. Tait, and C. Bloor. Evaluation of Information Seeking Performance in Hypermedia Digital Libraries. *Interacting with Computers*, 10(3):269-284, 1998.

[77] A. Schmidt, M. Kersten, M. Windhouwer, and F. Waas. Efficient Relational Storage and Retrieval of XML Documents. *In Proc. WebDB 2000*, pp. 47-52.

[78] J. Shanmugasundaram, K. Tufte, G. He, C. Zhang, D. DeWitt, and J. Naughton. Relational Databases for Querying XML Documents: Limitations and Opportunities. *In Proc. VLDB Conference 1999*.

[79] Techquila. TM4J A Free Topic Map Construction Tool. Available at http://www.techquila.com.

[80] TREC (Text REtrieval Conference) Home Page. Available at http://trec.nist.gov/.

[81] A. N. Vo, O. Krester, and A. Moffat. Vector-Space Ranking with Effective Early Termination. *In Proc. ACM SIGIR 2001*, pp. 35-42.

[82] I. H. Witten, A. Moffat, and T. C. Bell. *Managing Gigabytes: Compressing and Indexing Documents and Images*. Morgan Kaufmann, 1999.

[83] I. H. Witten, G. W. Paynter, E. Frank, C. Gutwin, and C. G. Nevill-Manning. KEA: Practical Automatic Keyphrase Extraction. *In Proc. Fourth ACM Conference on Digital Libraries 1999*, pp. 254-255.

[84] Extensible Markup Language (XML) 1.0, 1998. Available at http://www.w3.org/TR/REC-xml.

[85] Y. Yang, and M. Singhal. A Comprehensive Survey of Join Techniques in Relational Databases. Technical Report. Ohio State University, Computer and Information Science, 1997.

Appendix A

Extended SQL Queries Used in Experiments

The Appendix includes the extended SQL statements of the queries that were run over the Stephen King metadata database during the performance evaluation experiments of our metadata-based Web querying system.

A.1 Queries Involving SVA Operators

In this section, we provide the extended SQL statements and the logical query trees for the queries that include at least one SVA operator and were run over the Stephen King metadata database for the first part of the performance experiments (see Section 6.3.1).

S.King Query 1: (*SVA Selection*) Using the advice at www.Stephen-King.com/advice, find the names and URLs of 25 highest topic importance ranked novels having titles most similar to the string "dark tower". Return only the URLs of the topic sources of type "bibliography", and employ a product based topic importance propagation function.

select T.TName, S.URL
using advice at www.StephenKing.com/advice **as database** DB
from DB.Topics T, DB.TSRef S
where T.TType="Novel" **and**
T.TName ≅ "Dark Tower" **and**
T.TId **in** S.TId **and**
S.Role="bibliography"
propagate importance as product **function of** T
stop after 25 **most important**

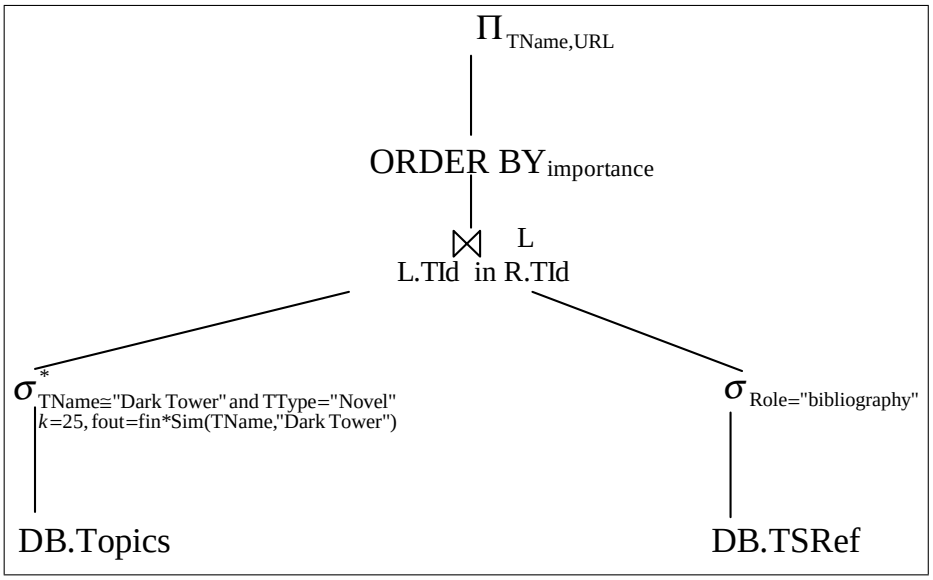

Figure A.1: Query tree for S. King query 1

S.King Query 2: (*SVA Directional Join*) Using the advice at www.StephenKing.com/advice, for each novel written by Stephen King, find 5 movies or tv-films having titles most similar to the title of the novel. Return their sources of type "summary", and employ a product based topic importance propagation function.

select T1.TName, S1.URL, T2.TName, T2.URL
using advice at www.StephenKing.com/advice **as database** DB
from DB.Topics T1, DB.Topics T2, DB.Topics T3,

DB.TSRef S1, DB.TSRef S2, DB.WrittenBy M
where T1.TType = "Novel" **and**
(T2.TType ="movie" **or** T2.TType ="tv-film") **and**
T3.TType= "Author" **and**
T3.TName= "Stephen King" **and**
T3.TId = M.Ant-Id **and** T1.TId = M.Cons-Id **and**
T1.TName $\cong_{(dir,k=5)}$ T2.TName **and**
T1.TId **in** S1.TId **and** T2.TId **in** S2.TId **and**
S1.Role="summary" **and** S2.Role="summary"
propagate importance as product **function of** T1, T2

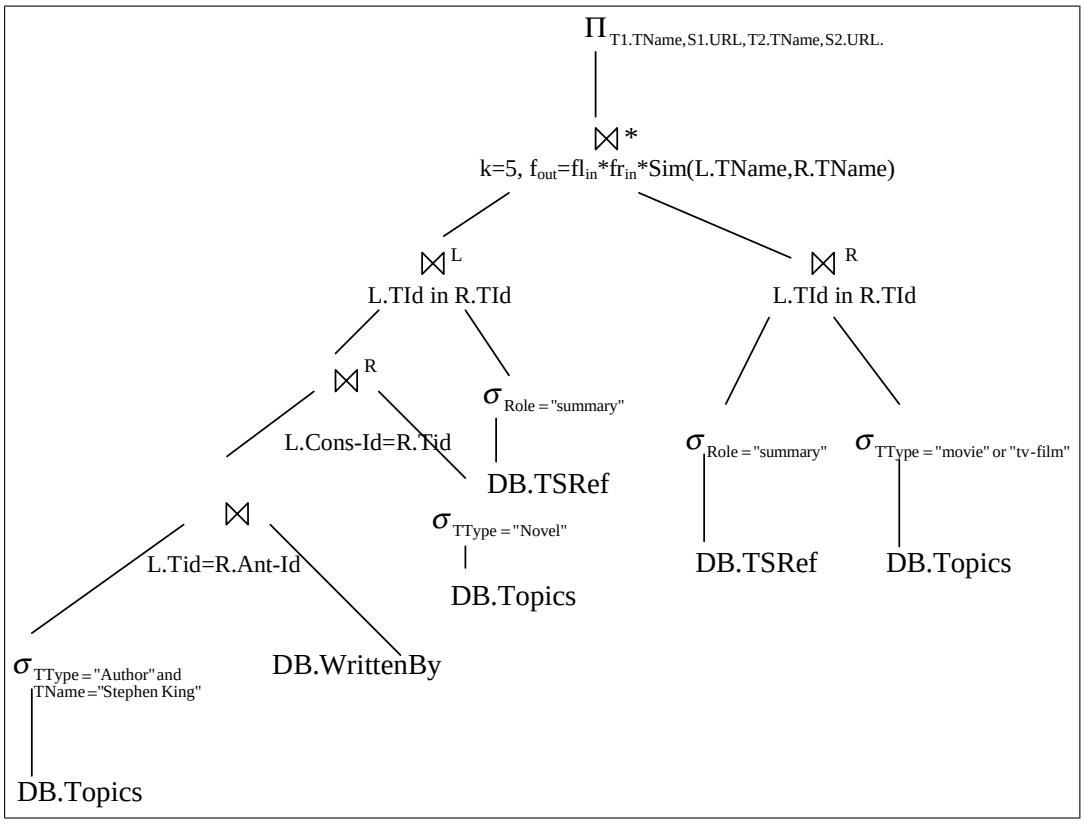

Figure A.2: Query tree for S. King query 2

S.King Query 3: (*SVA Topic Closure*) Using the advice at www.Stephen-King.com/advice, find the titles and URLs of 20 highest importance-valued novels such that the selected novels are prerequisite to the novel having the title "Wizard & Glass". Return their "description" type of sources.

select T2.TName, S2.URL
using advice at www.StephenKing.com/advice **as database** DB
from DB.Topics T, DB.Topics T1, DB.Topics T2, DB.Prerequisite M,
DB.TSRef S2
where T1.TName= "Wizard & Glass" **and**
T1.TType= "Novel" **and**
T2.TId **in** *Prerequisite**(T1.TId, T, M) **and**
T2.TId **in** S2.TId **and**
S2.Role="description"
topic closure importance computation as
product **function within a path and as**
max **function among multiple paths**
stop after 20 **most important**

S.King Query 4: (*User Profile*) Using the advice at www.StephenKing.com/advice, and the user profile at www.StephenKing.com/user, find the titles and URLs of 10 highest importance-valued novels such that the selected novels are related to the novel having the title "Night Journey". Return their "description" type of sources. Assume that the user wants to see topics having the string "Green Mile" in the query output.

select T2.TName, S2.URL
using advice at www.StephenKing.com/advice **as database** DB
using profile at www.StephenKing.com/user **as database** U
from DB.Topics T, DB.Topics T1, DB.Topics T2, DB.RelatedTo M,
DB.TSRef S2
where T1.TName= "Night Journey" **and**
T1.TType= "Novel" **and**
T2.TId **in** *RelatedTo**(T1.TId, T, M) **and**
T2.TId **in** S2.TId **and**
S2.Role="description"
topic closure importance computation as
product **function within a path and as**

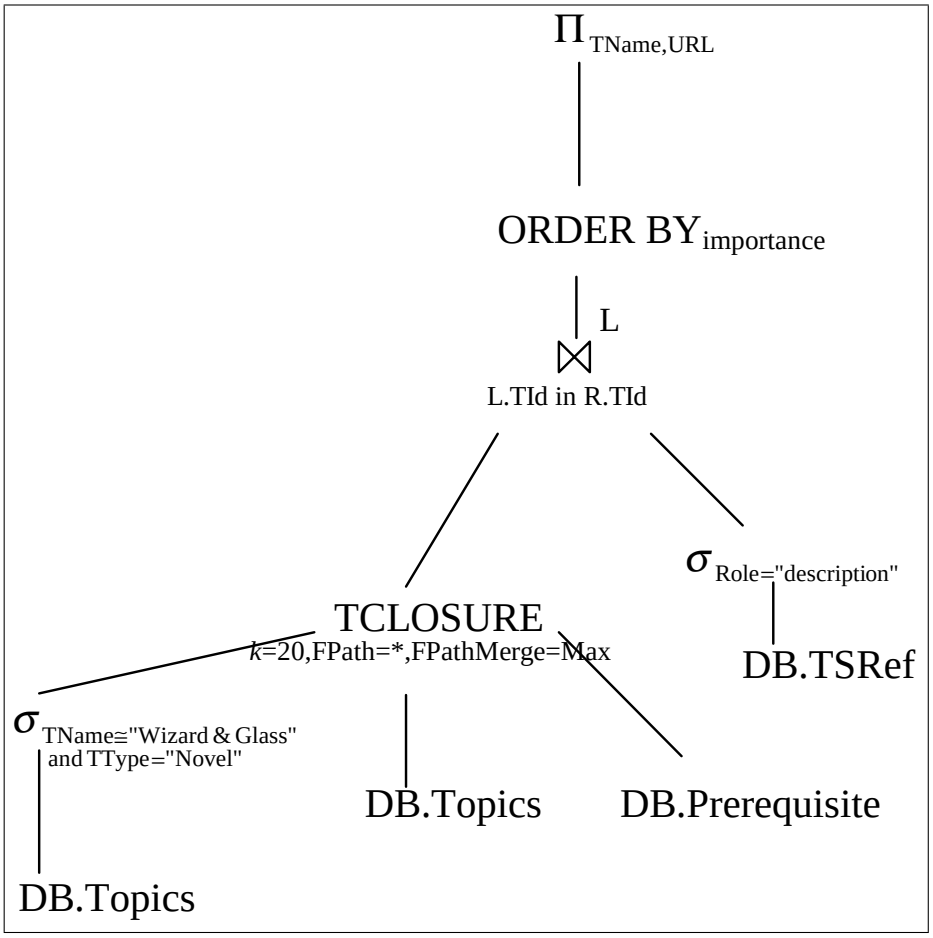

Figure A.3: Query tree for S. King query 3

max **function among multiple paths**
stop after 10 **most important**

S.King Query 5: Using the advice at www.StephenKing.com/advice, find the titles and URLs of 10 highest importance-valued novels such that the selected novels are related to the novel having the title "Night Journey". Return their "description" type of sources.

select T2.TName, S2.URL
using advice at www.StephenKing.com/advice **as database** DB
from DB.Topics T, DB.Topics T1, DB.Topics T2, DB.RelatedTo M,
DB.TSRef S2

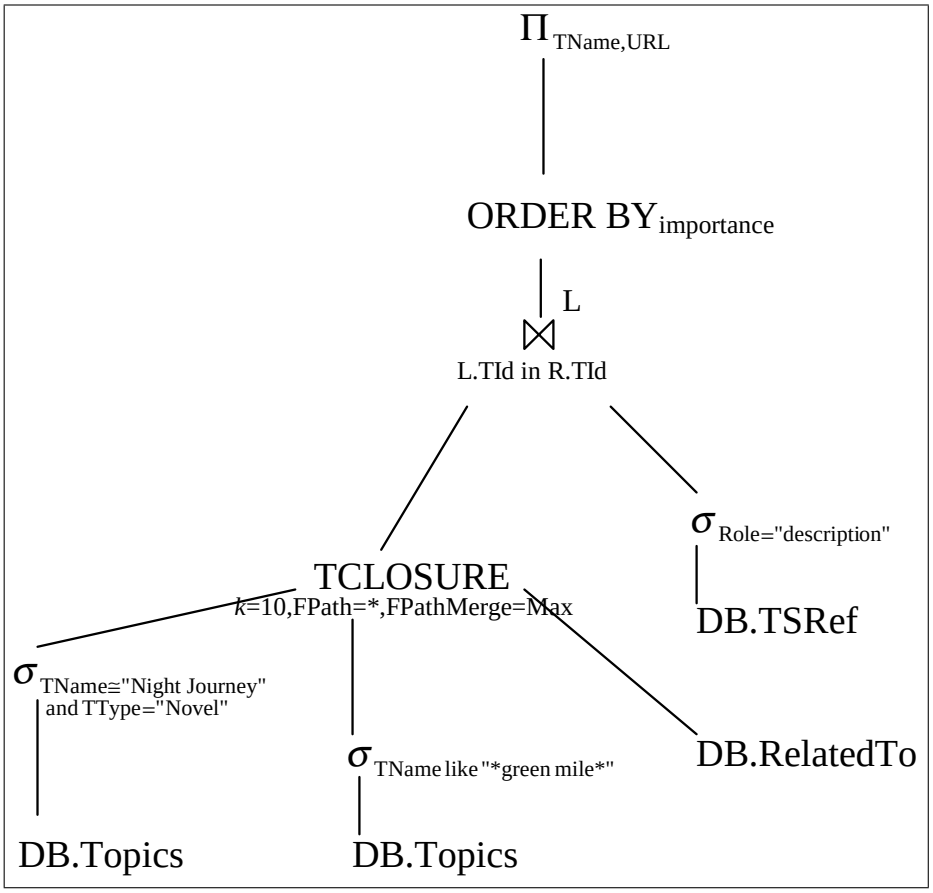

Figure A.4: Query tree for S. King query 4

where T1.TName= "Night Journey" **and**
T1.TType= "Novel" **and**
T2.TId **in** *RelatedTo**(T1.TId, T, M) **and**
T2.TId **in** S2.TId **and**
S2.Role="description"
topic closure importance computation as
product **function within a path and as**
max **function among multiple paths**
stop after 10 **most important**

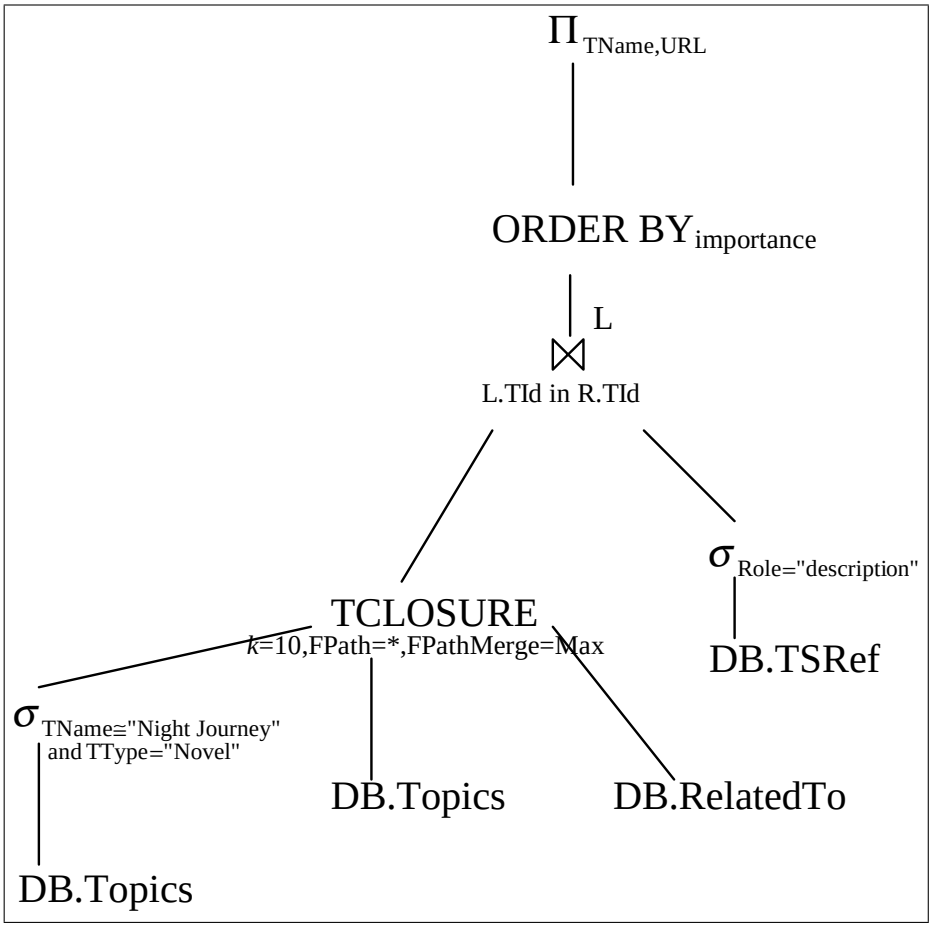

Figure A.5: Query tree for S. King query 5

A.2 Queries Not Involving SVA Operators

This section presents the extended SQL statements of the queries that do not include any SVA operator and were run over the Stephen King metadata database for the second part of the performance experiments (see Section 6.3.2). The queries given in this section can also be formulated as keyword queries.

S. King Query 1: Find all novels written by Stephen King.

select T.TName, S.URL
using advice at www.StephenKing.com/advice **as database** DB
from DB.Topics T, DB.Topics T1, DB.WrittenBy M, DB.TSRef S
where T.TId=M.Cons-Id **and**

T1.TName="Stephen King" **and** T1.TType="Author" **and**
M.Ant-Id=T1.TId **and** T.TId **in** S.TId
order by S.S-advice **desc**

S. King Query 2: Find reviews for the novel "Carrie".

select T.TName, S.URL
using advice at www.StephenKing.com/advice **as database** DB
from DB.Topics T, DB.TSRef S
where T.TName="Carrie" **and** T.TType = "Novel" **and**
T.TId **in** S.TId **and** S.Role = "review"
order by S.S-advice **desc**

S. King Query 3: Find biography of Stephen King.

select T.TName, S.URL
using advice at www.StephenKing.com/advice **as database** DB
from DB.Topics T, DB.TSRef S
where T.TName="Stephen King" **and** T.TType = "Author" **and**
T.TId **in** S.TId **and** S.Role = "biography"
order by S.S-advice **desc**

S. King Query 4: Find the list of Stephen King books published in year 1999.

select T.TName, S.URL
using advice at www.StephenKing.com/advice **as database** DB
from DB.Topics T, DB.PublicationYear M, DB.WrittenBy M2,
DB.Topics T1, DB.Topics T2, DB.TSRef S
where T.TId=M.Ant-Id **and**
T1.TName=1999 **and** T1.TId=M.Cons-Id **and**
T.TId=M2.Cons-Id **and** T2.TId=M2.Ant-Id **and**
T2.TName="Stephen King" **and** T.TId **in** S.TId

S. King Query 5: Find commercial sites for the novel "Dark Half".

```
select T.TName, S.URL
using advice at www.StephenKing.com/advice as database DB
from DB.Topics T, DB.TSRef S
where T.TName="Dark Half" and T.TType="Novel" and
        T.TId in S.TId and S.Role="commercial"
order by S.S-advice desc
```

S. King Query 6: Find all novels of Stephen King, which are not published by "Viking".

```
select T.TName, S.URL
using advice at www.StephenKing.com/advice as database DB
from DB.Topics T, DB.PublisherOf M, DB.WrittenBy M2,
      DB.Topics T1, DB.TSRef S
where T.TId=M.Ant-Id and
        M.Cons-Id not in (select T1.TId
                          from DB.Topics T1
                          where T1.TName="Viking" and
                                T1.TType="Publisher") and
        T.TId=M2.Cons-Id and T1.TId=M2.Ant-Id and
        T1.TName="Stephen King" and T.TId in S.TId
order by S.S-advice desc
```

S. King Query 7: Find the latest work of Stephen King.

```
select T.TName, S.URL
using advice at www.StephenKing.com/advice as database DB
from DB.Topics T, DB.PublicationYear M, DB.WrittenBy M2,
      DB.Topics T1, DB.TSRef S
where T.TId=M.Ant-Id and
```

M.Cons-Id **in** (**select** max(T1.TId)
from DB.Topics T1
where T1.TType="Year-Id") **and**
T.TId=M2.Cons-Id **and** T1.TId=M2.Ant-Id **and**
T1.TName="Stephen King" **and** T.TId **in** S.TId
order by S.S-advice **desc**

S. King Query 8: Find the description or summaries of all movies and tv-films based on the novel "Dark Half".

select T.TName, S.URL
using advice at www.StephenKing.com/advice **as database** DB
from DB.Topics T, DB.Topics T1, DB.BasedOn M, DB.TSRef S
where T1.TName="Dark Half" **and** T1.TType="Novel" **and**
(T.TType="Movie" **or** T.TType ="Tv-film") **and**
T.TId=M.Cons-Id **and** T1.TId=M.Ant-Id **and**
T.TId **in** S.TId **and**
(S.Role="description" **or** S.Role="summary")
order by S.S-advice **desc**

S. King Query 9: Find the description or summaries of all movies and tv-films based on the novel "Night Shift".

select T.TName, S.URL
using advice at www.StephenKing.com/advice **as database** DB
from DB.Topics T, DB.Topics T1, DB.BasedOn M, DB.TSRef S
where T1.TName="Night Shift" **and** T1.TType="Novel" **and**
(T.TType="Movie" **or** T.TType ="Tv-film") **and**
T.TId=M.Cons-Id **and** T1.TId=M.Ant-Id **and**
T.TId **in** S.TId **and**
(S.Role="description" **or** S.Role="summary")
order by S.S-advice **desc**

S. King Query 10: Find the publication year and the publisher of the book "Dead Zone".

select T.TName, T1.TName, T2.TName
using advice at www.StephenKing.com/advice **as database** DB
from DB.Topics T, DB.Topics T1, DB.Topics T2
DB.PublisherOf M, DB.PublicationYear M2
where T.TName="Dead Zone" **and** T.TType="Novel" **and**
T.TId=M.Ant-Id **and** T1.TId=M.Cons-Id **and**
T.TId=M2.Ant-Id **and** T2.TId=M.Cons-Id

S. King Query 11: Find the summary and characters of the book "Dreamcatcher".

select T.TName, S.URL
using advice at www.StephenKing.com/advice **as database** DB
from DB.Topics T, DB.TSRef S
where T.TName="Dreamcatcher" **and** T1.TType="Novel" **and**
T.TId **in** S.TId **and**
(S.Role="character" **or** S.Role="summary")
order by S.S-advice **desc**

Extended SQL statements for the queries that do not include any SVA operator and were run over the DBLP Bibliography metadata database are very similar to the above statements.